WAYNE STATE UNIVERSITY

Multi-Layer Perceptron

Dr. Ratna Babu Chinnam

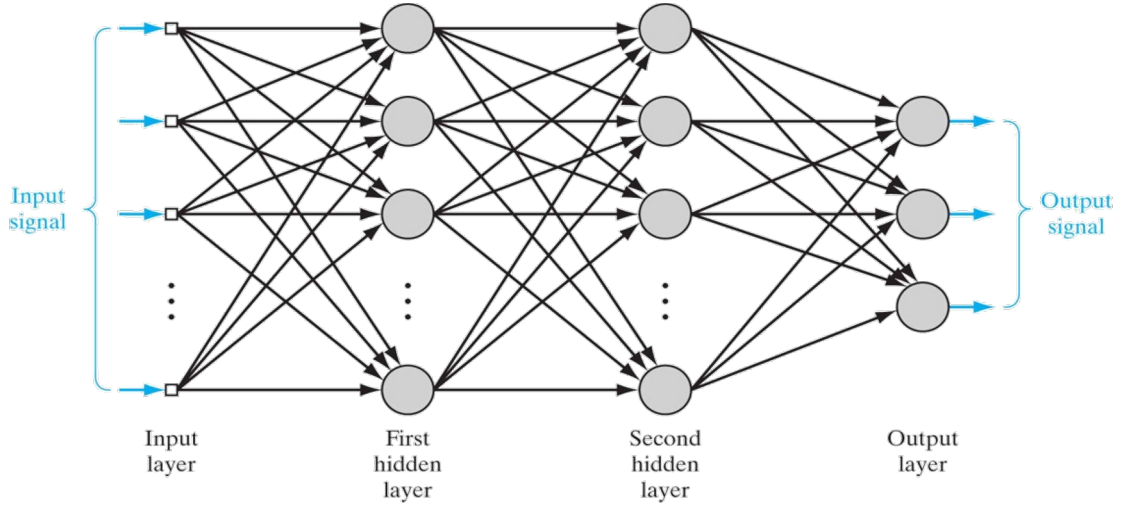
Industrial & Systems Engineering

Wayne State University

Primary Source: Simon Haykin (2009)

Multi-Layer Perceptron (MLP)

- Consists of ***input layer***, 1 or more ***hidden layers***, and ***output layer***



"Fully connected" typically implies that nodes in adjacent layers (not across layers) are fully connected!

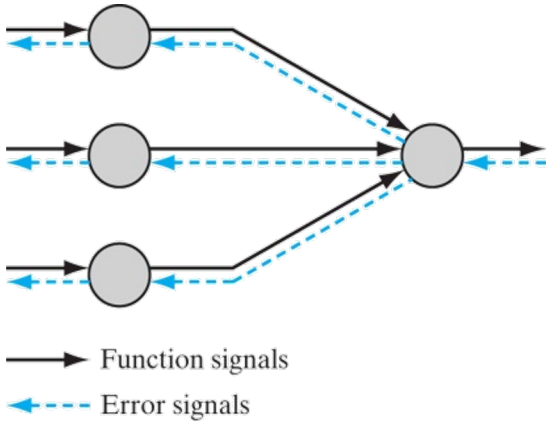
MLP is a *Universal Approximator*!

Architectural Graph of an MLP Network with Two Hidden Layers

- MLP networks are ***feed-forward networks*** (FFNs)
  - Signal propagates forward, layer-by-layer
- ***Trained in a supervised fashion*** using ***error back-propagation algorithm*** or its variants (based on error-correction rule)

Multi-Layer Perceptron

- ***Back-propagation learning*** consists of ***two passes:***
  - *Forward Pass:* Input signal propagates forward
  - *Backward Pass:* Error signal back-propagates

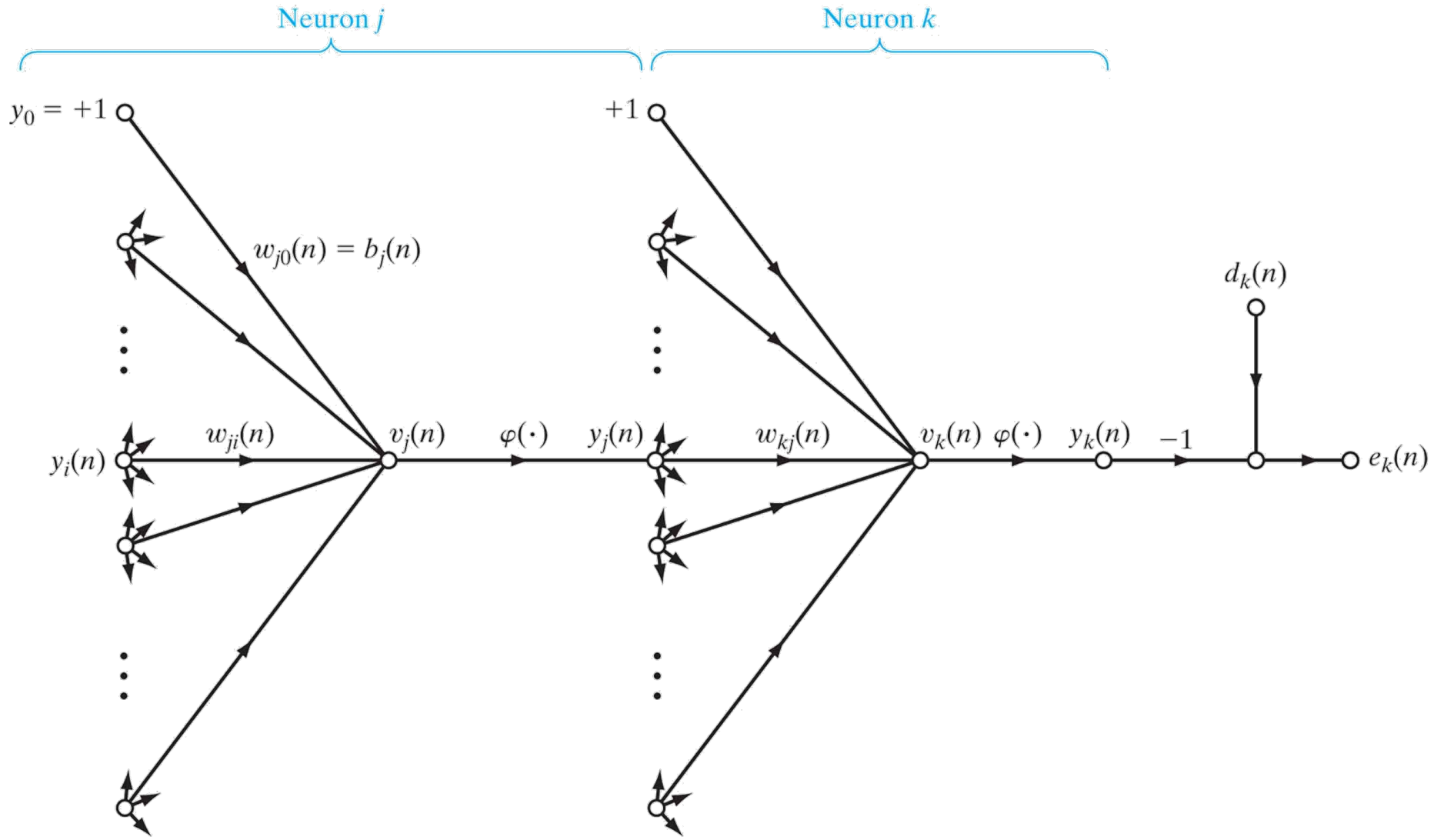


- **Three distinctive characteristics**:
  1. Neuron has a ***differentiable nonlinear activation function***
  2. Network contains 1 or more ***hidden layers***
  3. Network exhibits a high degree of ***connectivity***
- Development of ***back-propagation algorithm*** represents a landmark in that it provides a ***computationally efficient*** method for ***implementing the error-correction rule***

Back-Propagation: Notation

- Indices i, j, and k refer to different neurons in consecutive layers
- $\xi(n)$ denotes "instantaneous" sum of error squares at iteration n
- $y_k(n)$ and $d_k(n)$ denote actual and desired responses, at output neuron k
- $e_k(n)$ denotes neuron k error for iteration n: $e_k(n) = (d_k(n) - y_k(n))$
- $w_{ji}(n)$ denotes weight connecting neuron i to neuron j at iteration n
- $v_j(n)$ denotes activation potential of neuron j at iteration n
- $\varphi_j(\cdot)$ denotes activation function associated with neuron j
- b_j denotes bias for neuron j; represented by $w_{j0} = b_j$ and input ± 1
- η denotes learning rate parameter
- m_l denotes number of nodes in layer l of network; $l = 0,1,\ldots,L$, where L denotes number of hidden and output layers in network
- N denotes total number of patterns (data points) in training set

Back-Propagation: Notation



Signal-flow Graph Highlighting the Details of Neuron k Connected to Neuron j

Back-Propagation: Algorithm

- **Errors**: *Instantaneous* and *averaged* sum of squared errors for network

$$\xi(n) = \frac{1}{2}\sum_k e_k(n)^2 \qquad \xi_{av} = \frac{1}{N}\sum_{n=1}^{N}\xi(n)$$

- **Objective**: Adjust free parameters of network to minimize ξ_{av}
- Let us adjust weights pattern-by-pattern (i.e., "pattern mode" learning)
  - Average of individual weight changes over training set estimate change from directly minimizing ξ_{av} (i.e., "batch mode" learning)
- **Back-Propagation Algorithm:** Employs steepest-descent

$$\Delta w_{kj}(n) \propto \partial\xi(n)/\partial w_{kj}(n)$$

- **Per *delta rule***:

$$\Delta w_{kj}(n) = -\eta\frac{\partial\xi(n)}{\partial w_{kj}(n)}$$

$$= \eta\left[-\frac{\partial\xi(n)}{\partial v_k(n)}\right]\frac{\partial v_k(n)}{\partial w_{kj}(n)} \quad \text{(by chain rule)}$$

$$= \eta\delta_k(n)y_j(n)$$

where $\delta_k(n)$ denotes ***local gradient*** of neuron k

Back-Propagation: Algorithm

Case I: Neuron k is an Output Node

- According to the "chain" rule:

$$\frac{\partial \xi(n)}{\partial w_{kj}(n)} = \frac{\partial \xi(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial w_{kj}(n)}$$

- From our notation:

$$\frac{\partial \xi(n)}{\partial e_k(n)} = e_k(n) \quad \frac{\partial e_k(n)}{\partial y_k(n)} = -1 \quad \frac{\partial y_k(n)}{\partial v_k(n)} = \varphi'_k(v_k(n)) \quad \frac{\partial v_k(n)}{\partial w_{kj}(n)} = y_j(n)$$

- For current case where neuron j is an output node:

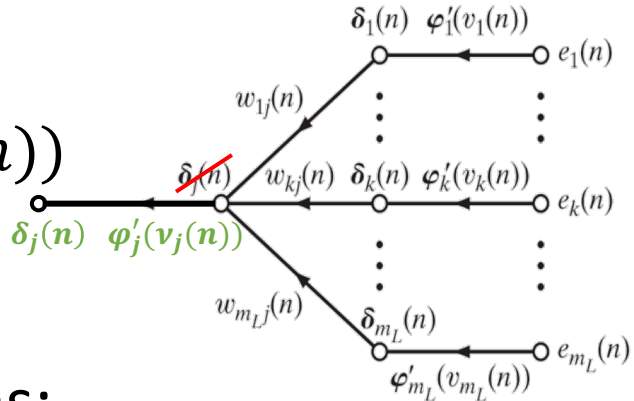
$$\delta_k(n) = -\frac{\partial \xi(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)}$$
$$= e_k(n) \varphi'_k(v_k(n))$$

Back-Propagation: Algorithm

Case II: Neuron *j* is a Hidden Node

- No specified desired response for neuron (hidden node)
  - Error to be determined in terms of errors of neurons to which it is connected
- Local gradient $\delta_j(n)$ for hidden neuron j is:

$$\delta_j(n) = -\frac{\partial \xi(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \xi(n)}{\partial y_j(n)} \varphi'_j(v_j(n))$$



- Instantaneous error: $\xi(n) = \frac{1}{2}\sum_k e_k(n)^2$
- Local gradient $\delta_j(n)$ for hidden neuron j can be rewritten as:

$$\delta_j(n) = -\varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \right] = -\varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \right]$$

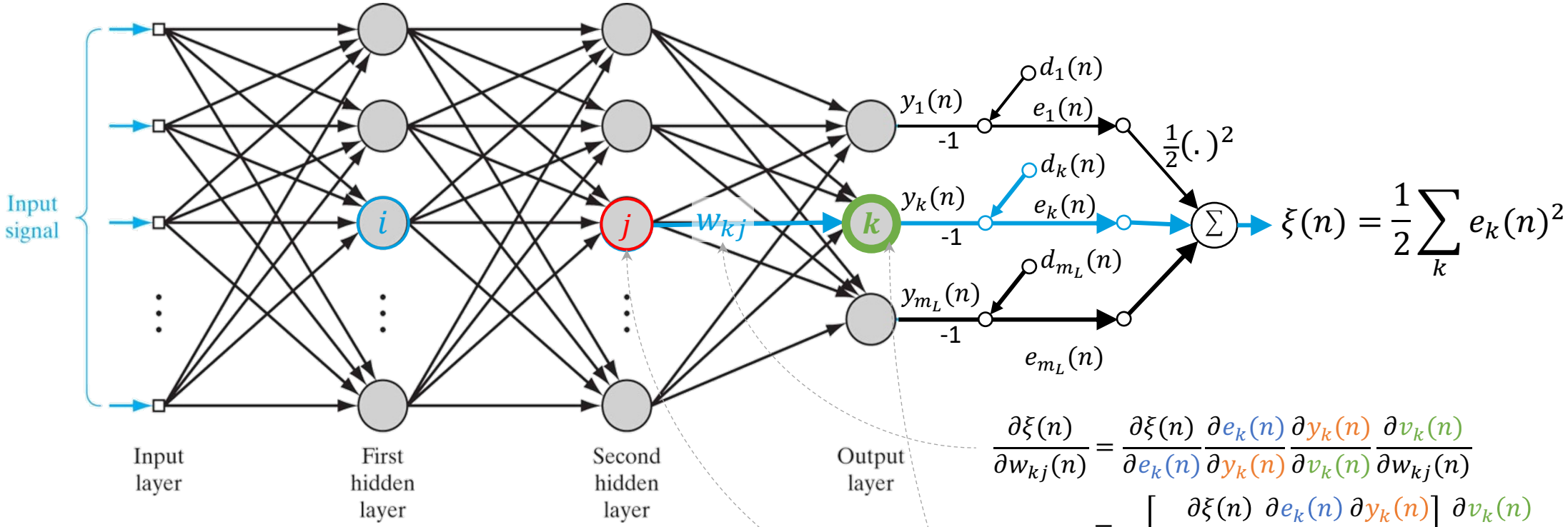
$$= -\varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n) \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \right] = -\varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n)(-1)\varphi'_k(v_k(n)) \frac{\partial v_k(n)}{\partial y_j(n)} \right]$$

$$= \varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n)\varphi'_k(v_k(n)) \frac{\partial \sum_j w_{kj}(n) y_j(n)}{\partial y_j(n)} \right] = \varphi'_j(v_j(n)) \cdot \sum_k [\delta_k(n) w_{kj}(n)]$$

- $\delta_j(n)$ can be shown to be valid for any neuron in any hidden layer

Back-Propagation: Visualization

NEURON "k" IN OUTPUT LAYER



$$\frac{\partial \xi(n)}{\partial w_{kj}(n)} = \frac{\partial \xi(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial w_{kj}(n)}$$

$$= -\left[-\frac{\partial \xi(n)}{\partial e_k(n)} \frac{\partial e_k(n)}{\partial y_k(n)} \frac{\partial y_k(n)}{\partial v_k(n)}\right] \frac{\partial v_k(n)}{\partial w_{kj}(n)}$$

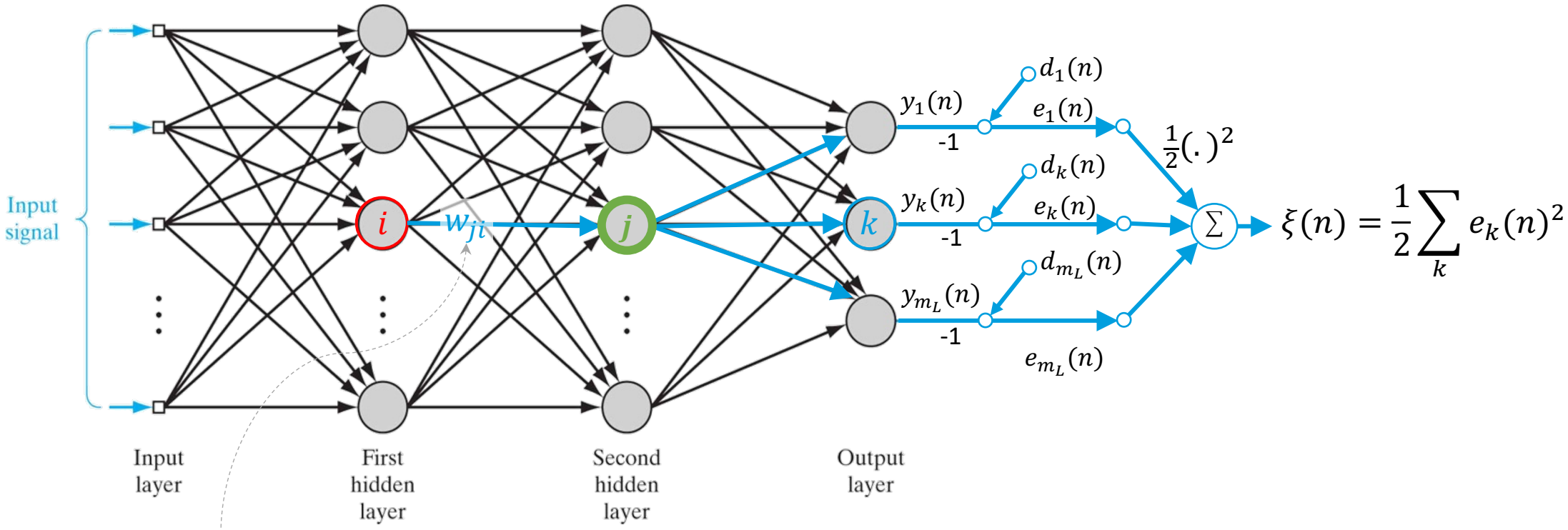
$$= -\left[-e_k(n)(-1)\,\varphi'_k(v_k(n))\right] y_j(n)$$

$$= -\left[e_k(n)\,\varphi'_k(v_k(n))\right] y_j(n)$$

$$= -\delta_k(n)\, y_j(n)$$

Back-Propagation: Visualization

NEURON 'j' IN LAST HIDDEN LAYER

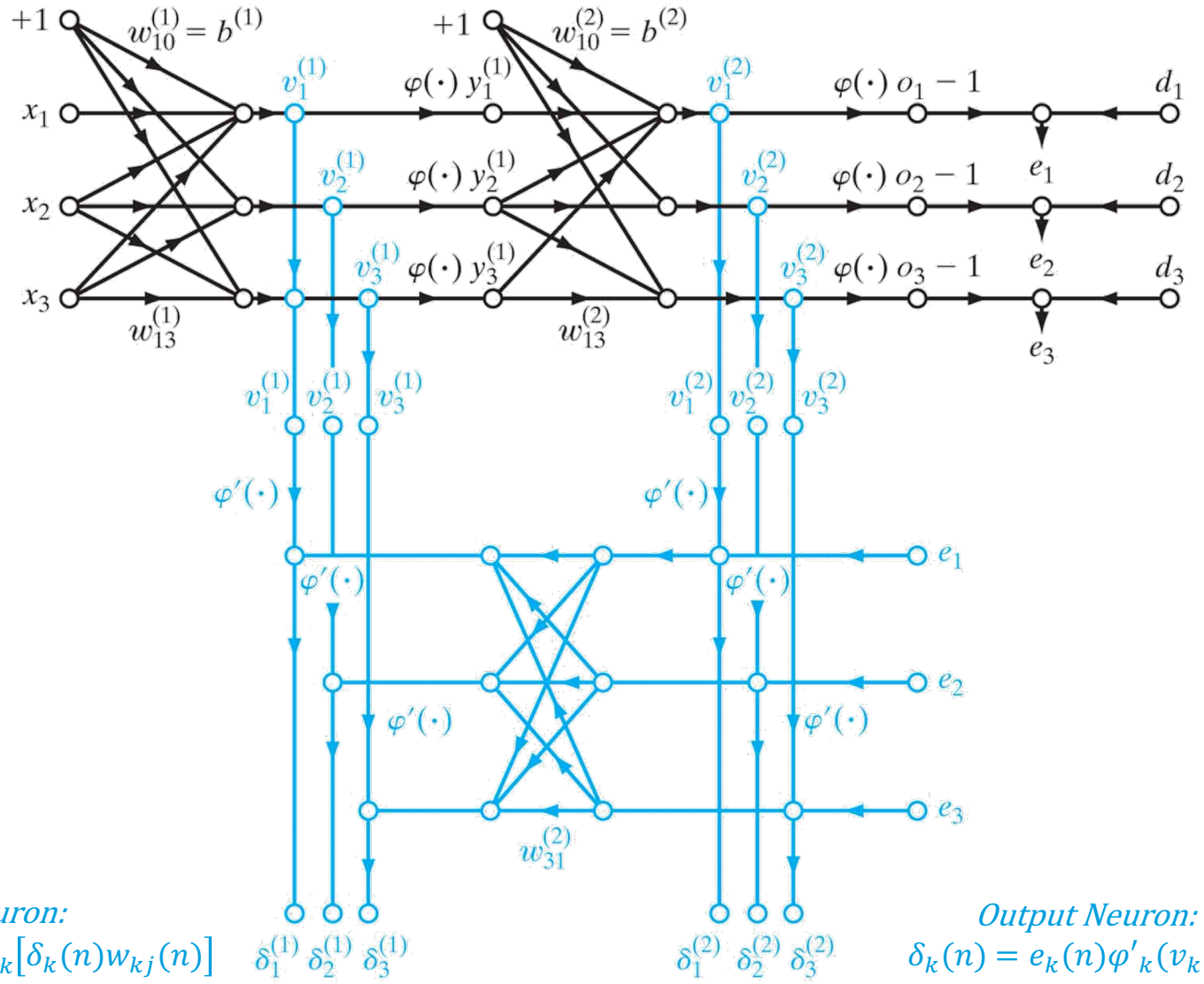


$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -\varphi'_j(v_j(n)) \cdot \sum_k \left[e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \right] y_i(n)$$

$$= -\left[\varphi'_j\left(v_j(n)\right) \cdot \sum_k \left[\delta_k(n) w_{kj}(n)\right] \right] y_i(n)$$

$$= -\delta_j(n)\, y_i(n)$$

Signal-flow Summary of Back-propagation Algorithm



Hidden Neuron:

$$\delta_j(n) = \varphi'_j(v_j(n)) \cdot \sum_k [\delta_k(n) w_{kj}(n)]$$

Output Neuron:

$$\delta_k(n) = e_k(n) \varphi'_k(v_k(n))$$

Representation: Top Part→Forward Pass. **Bottom Part→Backward Pass.**

Activation Function

- Local gradient requires derivative of activation function ($\varphi'(\cdot)$)
- ***Logistic Function:***

$$\varphi_j(v_j(n)) = \frac{a}{1+\exp(-bv_j(n))} \quad (a,b) > 0 \quad -\infty < v_j(n) < +\infty$$

$$\varphi'_j(v_j(n)) = \frac{ab\exp(-bv_j(n))}{\left[1+\exp(-bv_j(n))\right]^2} = \frac{b}{a}\varphi_j(v_j(n))\left[a-\varphi_j(v_j(n))\right]$$

- ***Hyperbolic Tangent Function:***

$$\varphi_j(v_j(n)) = a\cdot\tanh(bv_j(n)) \quad (a,b) > 0 \quad -\infty < v_j(n) < +\infty$$

$$= a\cdot\frac{1-\exp(-bv_j(n))}{1+\exp(-bv_j(n))}$$

$$\varphi'_j(v_j(n)) = ab\cdot\operatorname{sech}^2(bv_j(n))$$

$$= ab(1-\tanh^2(bv_j(n))) = \frac{b}{a}\left[a+\varphi_j(v_j(n))\right]\left[a-\varphi_j(v_j(n))\right]$$

Activation Function

- ***Softmax Function:***

$$p_j = \text{softmax}(\boldsymbol{y} = [y_1, \ldots, y_K]) = \frac{e^{y_j}}{\sum_{k=1}^{K} e^{y_k}}$$

$$\frac{\partial p_j(n)}{\partial y_i(n)} = \begin{cases} p_j(1 - p_i) & if\ j = i \\ -p_j \cdot p_i & if\ j \neq i \end{cases}$$

See Full Derivation Here: Link

- ***ReLu Functions:***

Rectifer:

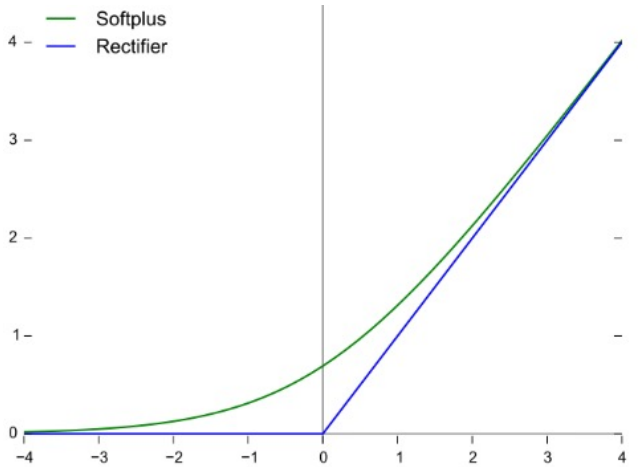
$$\varphi_j(v_j(n)) = \max(0, v_j(n))$$

$$\varphi'_j(v_j(n)) = \begin{cases} 1 & if\ v_j(n) > 0 \\ 0 & if\ v_j(n) \leq 0 \end{cases}$$

Softplus:

$$\varphi_j(v_j(n)) = \log(1 + e^{v_j(n)})$$

$$\varphi'_j(v_j(n)) = \frac{e^{v_j(n)}}{1 + e^{v_j(n)}}$$



Rate of Learning: "Momentum" and $\eta(n)$

- Back-propagation algorithm provides an "approximation" to steepest descent during pattern-mode learning: **Stochastic Gradient Descent**
- **Rate of Learning**:
  - Small η: Smooth trajectory attained at the cost of a slower rate of learning
  - Large η: Faster learning but there is a danger network may become unstable
- **Momentum**: One can increase rate of learning while avoiding instability:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \quad 0 \leq |\alpha| < 1$$

$$= \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t)$$

  - Current adjustment represents sum of an exponentially weighted time series
- Tends to *accelerate descent* in steady downhill directions and has a *stabilizing effect* in directions that oscillate in sign
- **Learning Rate**: Can be adapted as well: $\eta(n)$ Ideas?

Sequential (Pattern) & Batch Modes of Training

- **Sequential Mode**: Adjusts network parameters for each example

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

  - Search in the weight space tends to be *stochastic* avoiding limit cycles
  - Order of presentation of training examples has to be randomized
  - Reduced memory storage
  - Increased potential for avoiding local minima
- **Batch Mode**: Adjusts network parameters after presenting all examples
  - Local gradients have to be calculated after presentation of each example

$$\Delta w_{ji} = \frac{\eta}{N} \sum_{n=1}^{N} \delta_j(n) y_i(n)$$

  - Easier to establish theoretical conditions for algorithm convergence
- **Note**: Computed $\delta_j(n)$ and $y_i(n)$ tend to differ for the two modes of training since the weights are updated at different frequencies

Useful Criteria for “Stopping” Network Training

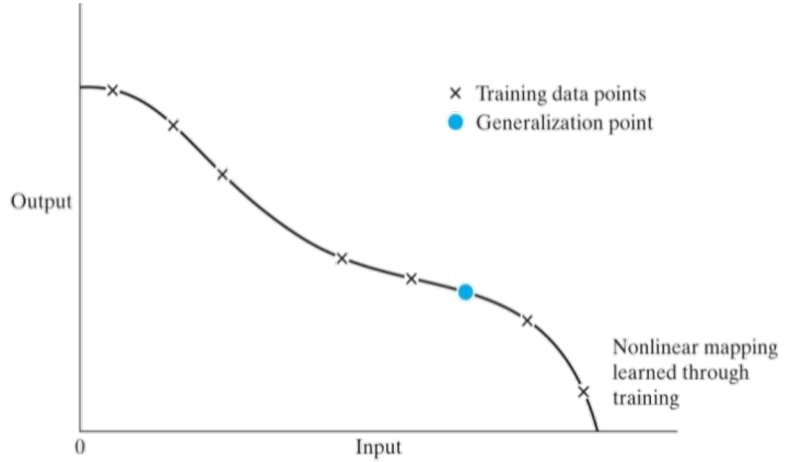
- **Epochs**: Predetermined number of “epochs”
  - Epoch: A single presentation of ALL training patterns to network
- **Computation Time**: Predetermined processor computation time
- **Learning Error**: Acceptable minimum learning error
- **Learning Rate**: Acceptable minimum learning rate
- **Generalization**:
  - Acceptable generalization performance
  - Apparent peak (actually, a “valley” in error plot) in generalization performance

“Heuristics” for Improving Back-propagation Algorithm

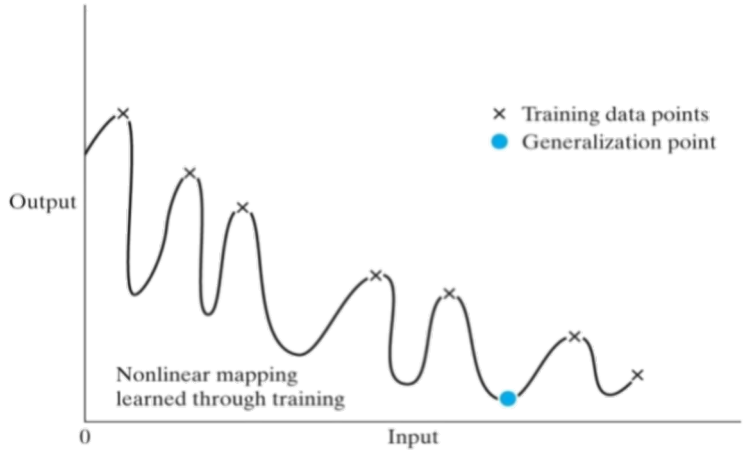
- ***Sequential versus batch update***. Sequential update leads to stochastic gradient search and can be more effective in practice
- ***Mini-batches provide a better balance*** between accuracy and computational efficiency
- ***Nature of activation function***. Antisymmetric functions generally help (i.e., $\varphi(-v) = -\varphi(v)$).
- ***Target values***. Linear output neurons might be safer for regression.
- ***Normalization of inputs and outputs***. Is crucial.
- ***Weight initialization***. Synaptic weights should be initialized uniformly with mean zero and variance equal to the reciprocal of the number of synaptic connections of a neuron.
- ***Learning rate parameters***. All neurons should learn at the same rate. Since the last layers have usually larger local gradients, their learning rate parameters should be smaller.
- ***A priori information***. Utilize this knowledge to achieve invariance properties, symmetries, etc.

Achieving “Generalization” with MLP

- Network is said to *generalize* well when the mapping is correct for test data
  - Assumption: Test data are drawn from the same process/population
- Curve-fitting Example: Equivalent to good interpolation
- *Overtraining* or *overfitting* can result if network starts “memorizing” training data
- Can be viewed from different perspectives:
  - Architecture is fixed, what should be the size of the training data?
    - VC dimension provides the theoretical basis
  - Training data set is fixed, what is the best architecture for the network?
    - More common approach



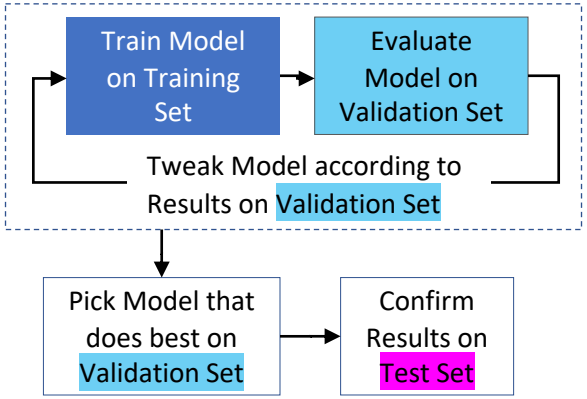
Mapping with good generalization

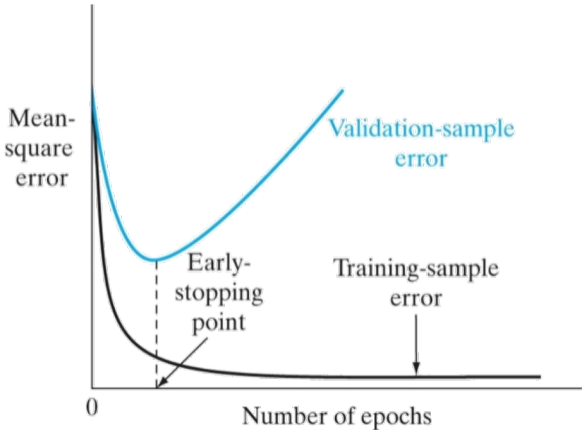


Mapping with poor generalization

Achieving Generalization through "Cross-Validation"

- "Training" can be seen as making the network "learn enough" to generalize
- **Cross-Validation**: A standard tool from statistics is valuable
  - Dataset is partitioned into ***training*** (N patterns) and ***test*** sets (M patterns)
  - Training set is further partitioned into two disjoint subsets:
    - ***Estimation*** subset to "learn" the model (with $(1-r)N$ patterns, where $0 < r < 1$)
    - ***Validation*** subset to "validate" the model (with rN patterns)
  - Possible for model to overfit validation subset. Measure performance on test set.
  - As target function becomes complex relative to N, choice of r^* becomes important and should decrease in value (Kearns, 1996)
- **Early Stopping:** A method of training using cross-validation
  - Periodically stop network training process and test network
  - Permanently stop training if an increase is noted in validation error over successive evaluations.
  - Effectiveness decreases if $N \gg W$





Variants of Cross-Validation

- For relatively small training data sets, two variants of cross-validation that offer more promise include:
- ***Multifold Method***
  - Involves dividing training set into K subsets, $K > 1$
  - Model is trained on all subsets except for one, and validation error is measured by testing it on the subset left out
  - Procedure is repeated for a total of K trials, each time using a different subset for validation
  - Model performance is assessed by averaging squared error under validation over all trials of the experiment
  - Increases computational burden
- ***Leave-one-out Method***
  - When data set is severely small, one can use extreme form of multifold cross-validation where K equals N

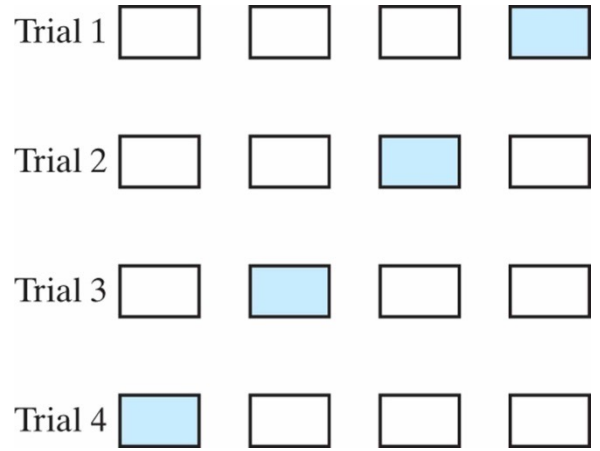
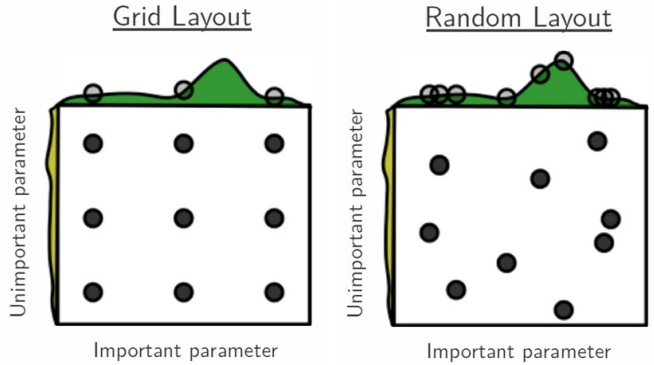


Illustration of the Hold-Out Method of Cross-Validation

Achieving Generalization: Hyper-Parameter Tuning

- **Need to optimize network structure and learning parameters**
  - Number of hidden layers
  - Nodes per layer
  - Transfer function/parameters
  - Learning rate/decay
  - Regularization parameters
  - Others …
- **Joint optimization is necessary**
  - Total Enumeration (Grid Search): Can be expensive
  - Random Search: Little more practical
  - Bayesian Optimization: Smart exploration & exploitation
- **Example Packages:**
  - Matlab: bayesopt | Link
  - Python: skopt | Link | Example; bayesopt | Example
  - Keras: Talos | Link
  - Google Cloud: Bayesian Optimization | Link



Achieving Generalization: Keras Code Options

Early Stopping using Keras Example: Documentation

```
keras.callbacks.EarlyStopping(monitor='val_loss',
min_delta=0, patience=0, verbose=0, mode='auto')
```

Regularization using Keras Example: Documentation

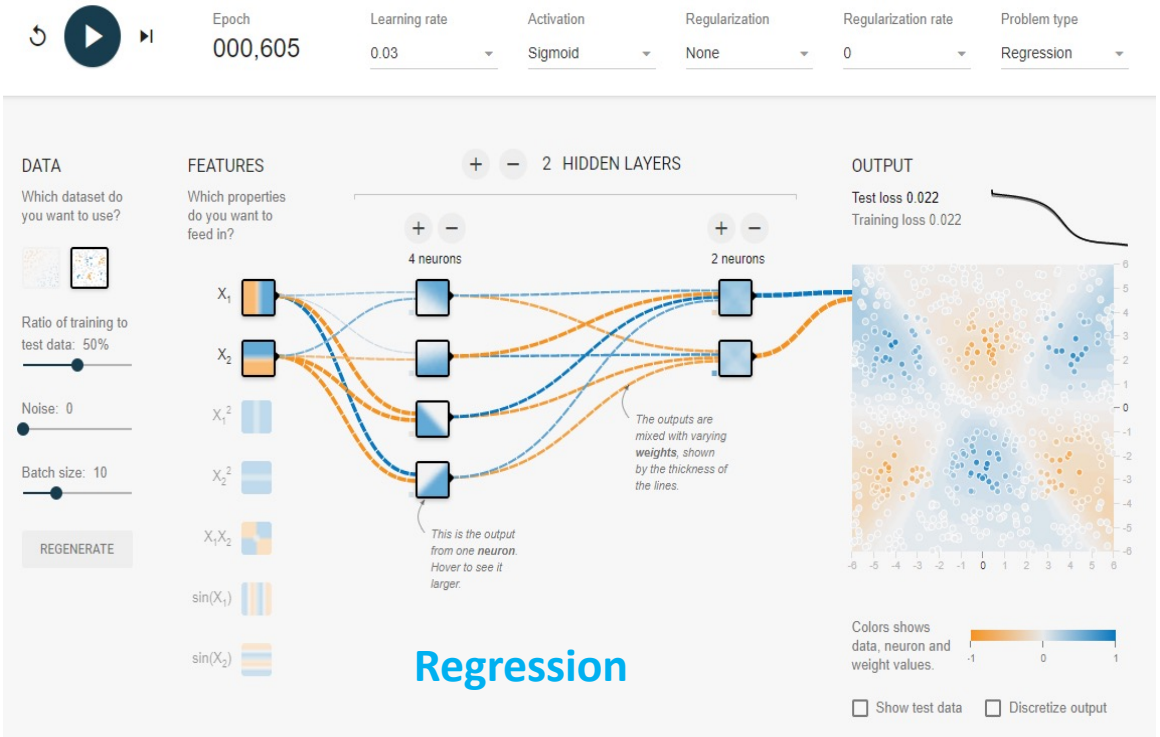
```
from keras import regularizers
model.add(Dense(64, input_dim=64,
kernel_regularizer=regularizers.l2(0.01),
activity_regularizer=regularizers.l1(0.01)))
```

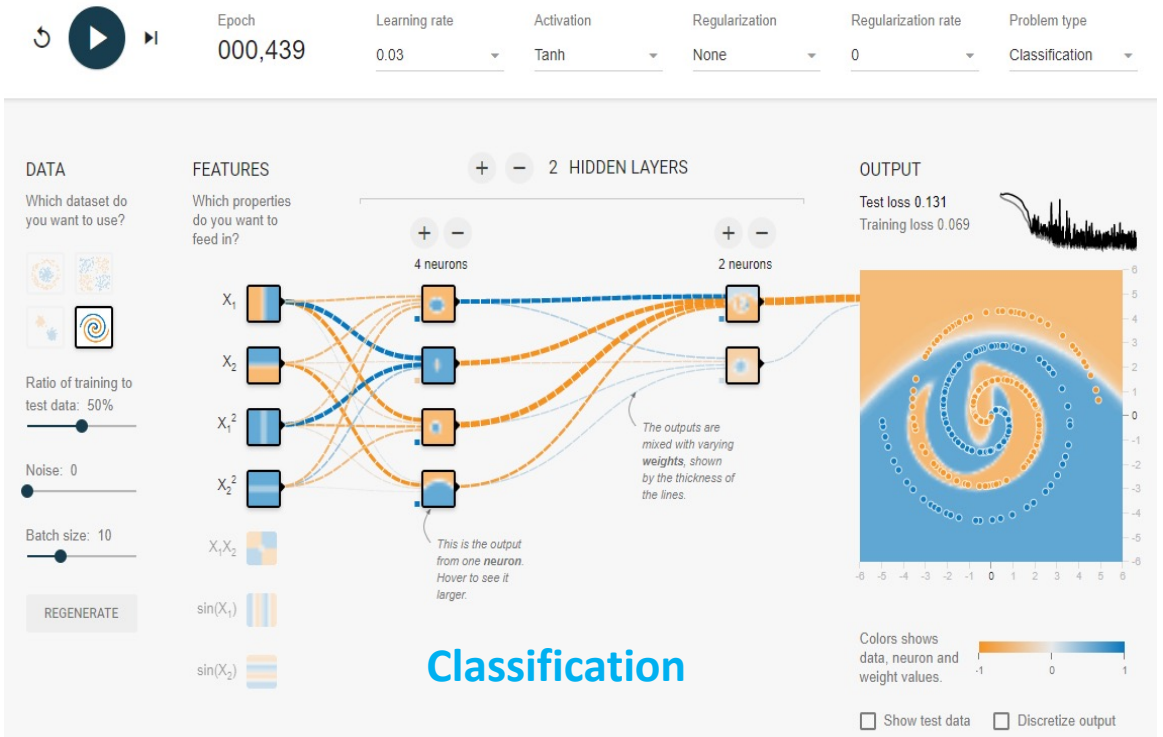
DropOut Layer using Keras Example: Documentation

```
keras.layers.Dropout(rate,
noise_shape=None, seed=None)
```

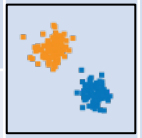
Neural Network Playground: Website

- Daniel Smilkov and Shan Carter created a [neural network playground] (using TensorFlow) to demystify MLP networks

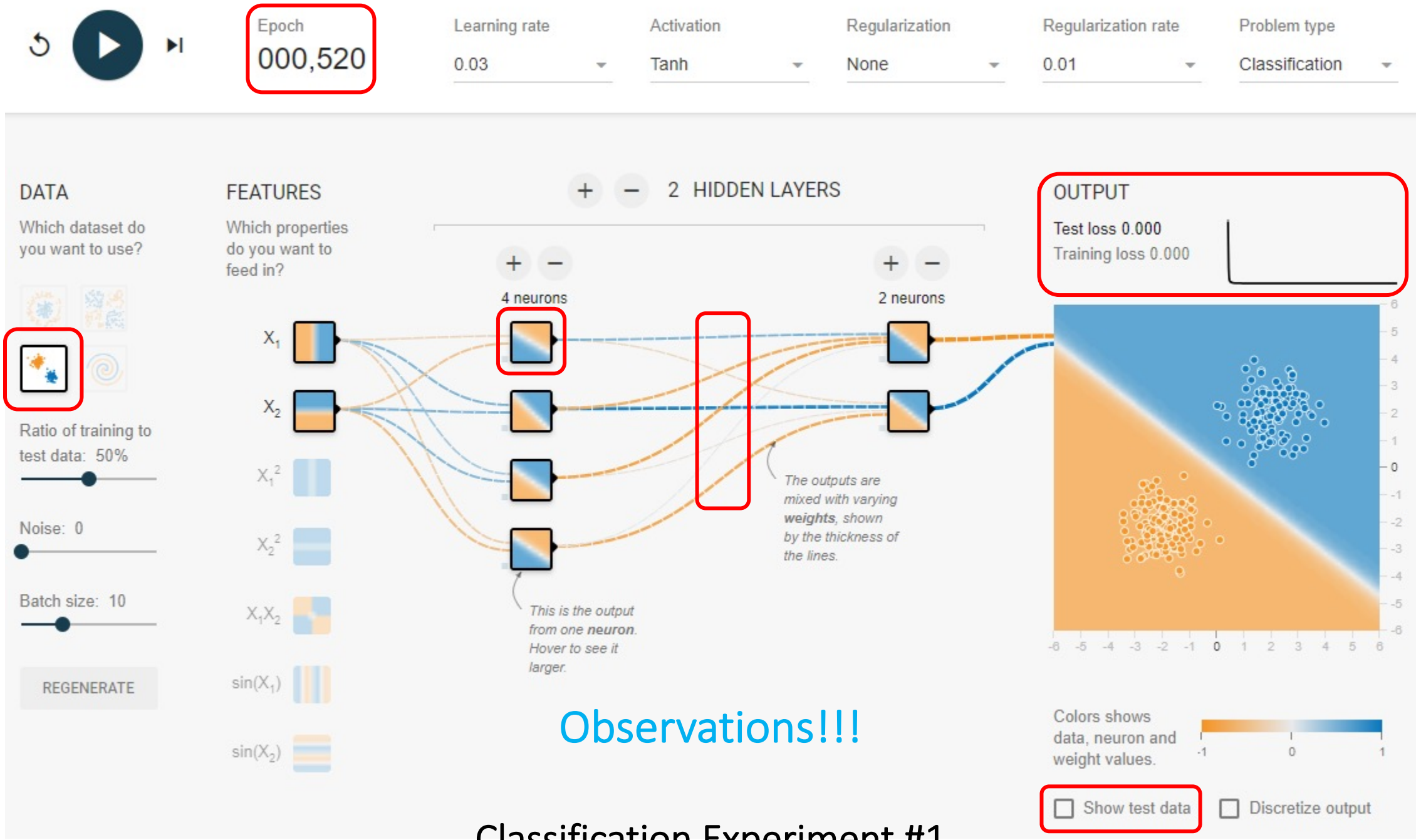




Neural Network Playground: Classification Experiments

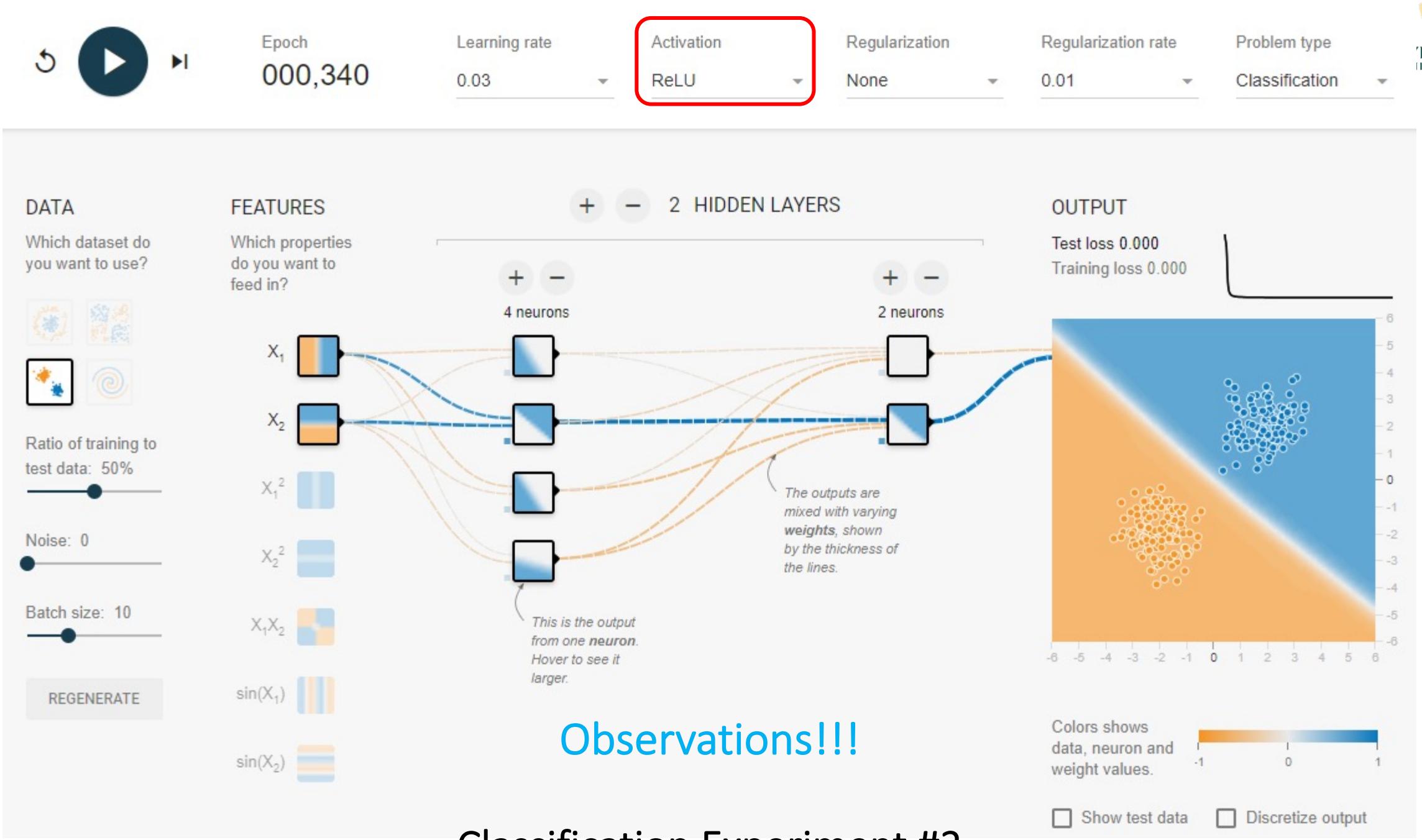
#	Data	Features	Hidden Layers & Nodes	Activation & Learning Rate	Regularization Type & Rate	OBSERVATIONS: Accuracy/Loss (Train/Test), Convergence Rate, Robustness, Others
1		X_1, X_2	2: 4, 2	Tanh, 0.03	None	
2				ReLU, 0.03		
3				Tanh, 0.03	L2, 0.01	
4					L1, 0.01	
5		$+X_1^2, X_2^2, X_1X_2$			None	
6					L2, 0.01	
7		X_1, X_2	2: 4, 2	Tanh, 0.03	None	
8			2: 8, 4			
9			3: 8, 4, 2			
10		$+ Sin(X_1), Sin(X_2)$	2: 4, 2			
11		X_1, X_2	5: 6,5,4,3,2			Batch Size: 30
12			3: 8, 5, 3			

Other Settings: Ratio of Training to Test Data=50%, Noise=0, Batch Size=10



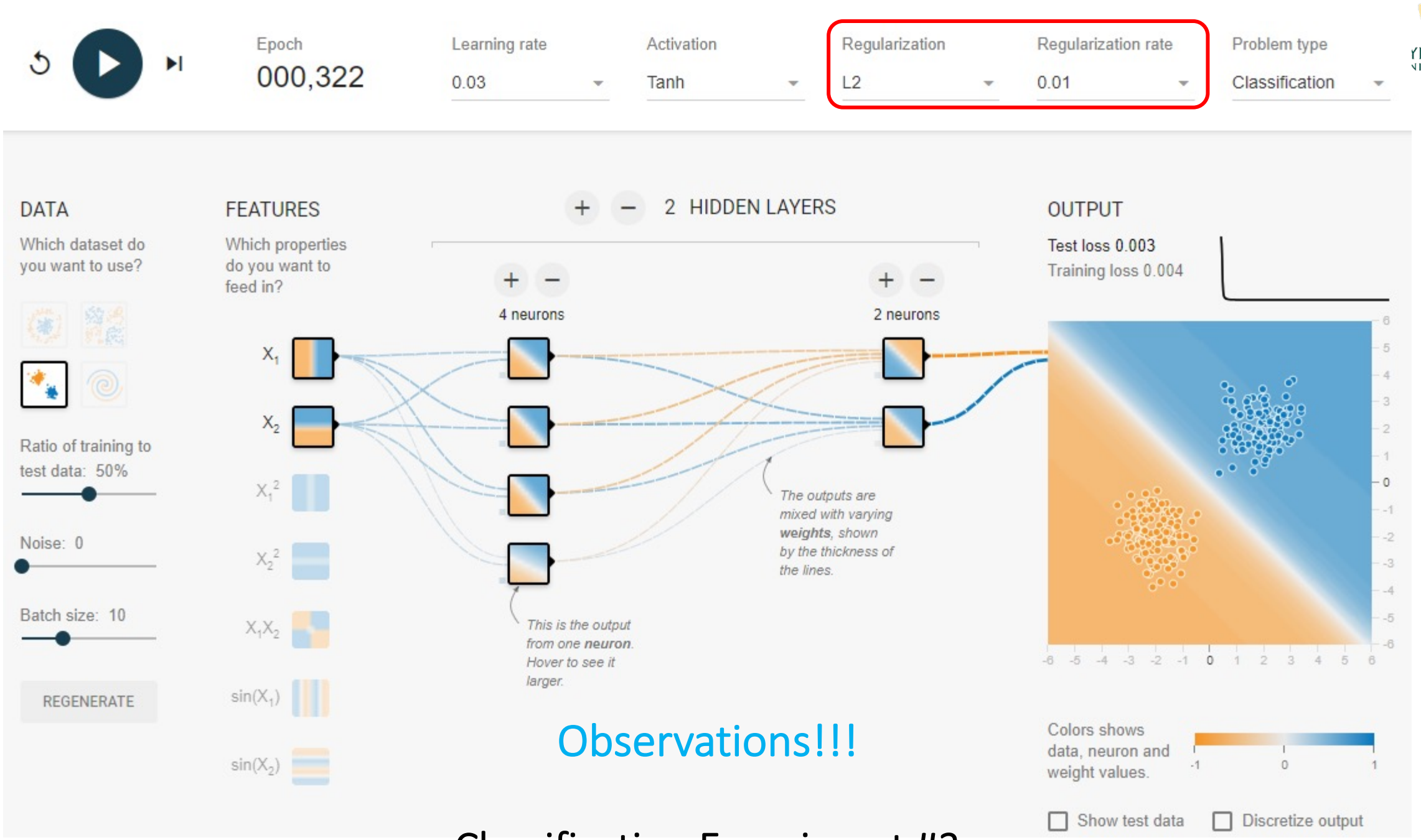
Observations!!!

Classification Experiment #1



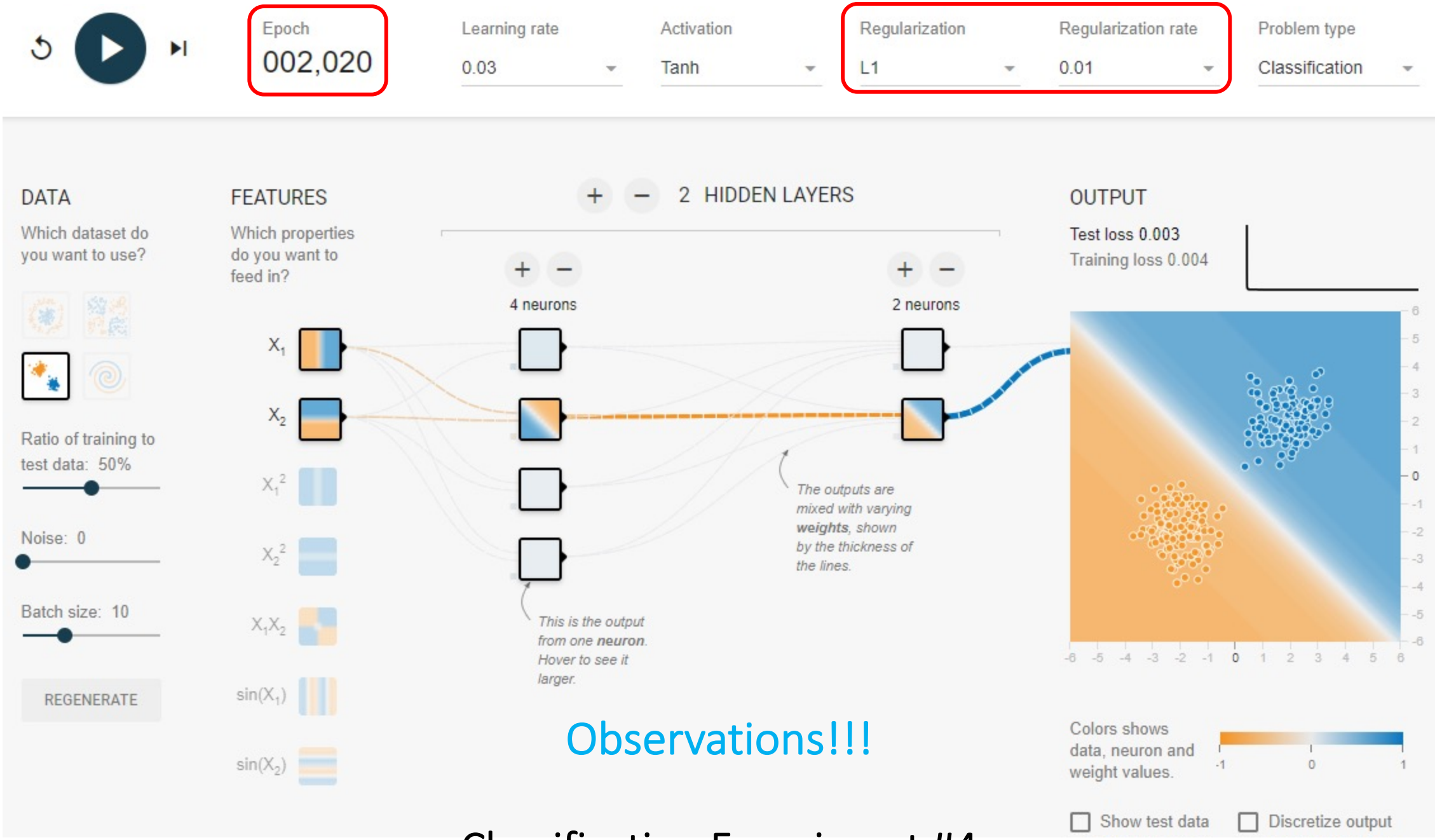
Observations!!!

Classification Experiment #2



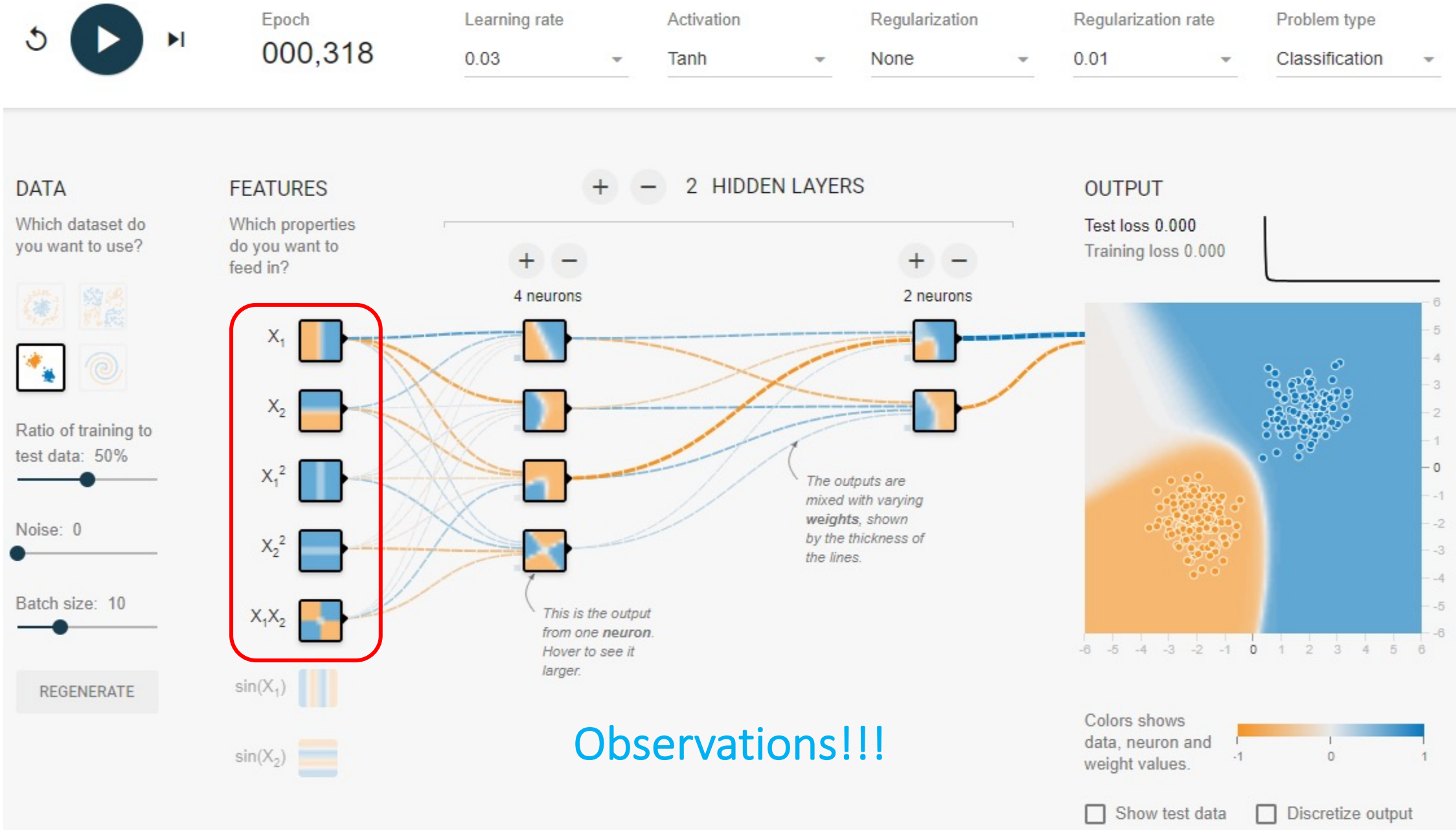
Observations!!!

Classification Experiment #3

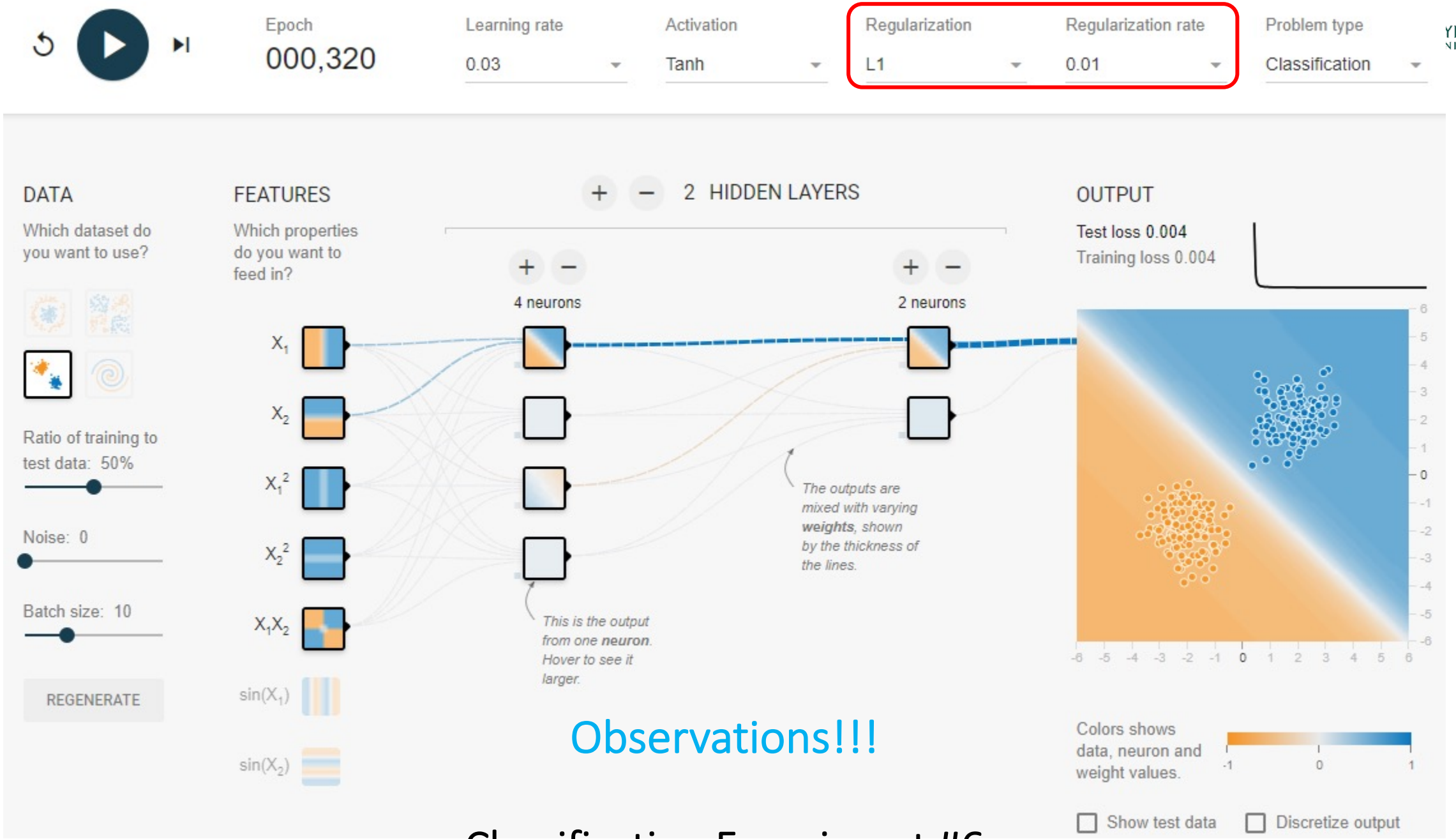


Observations!!!

Classification Experiment #4

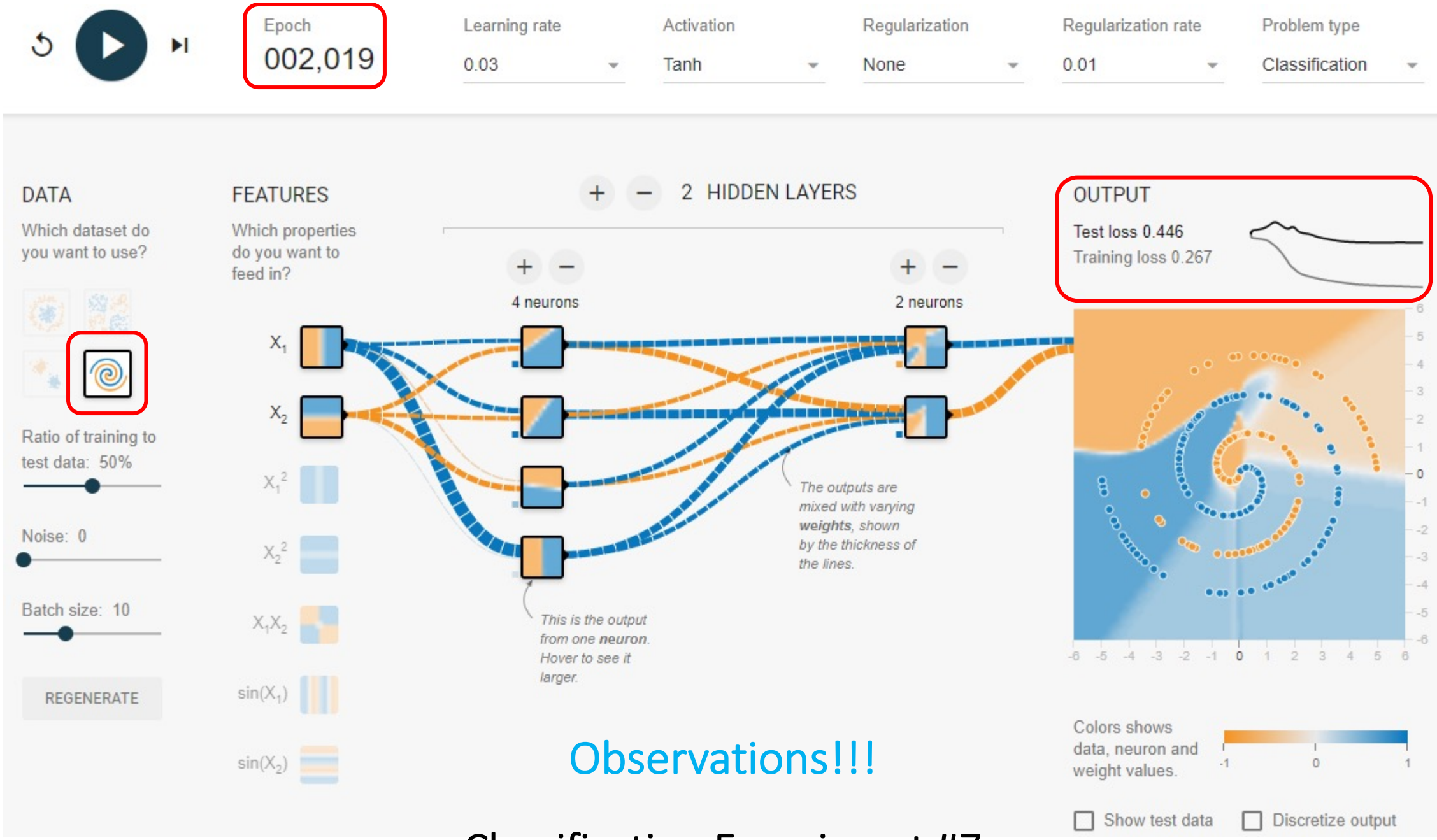


Classification Experiment #5



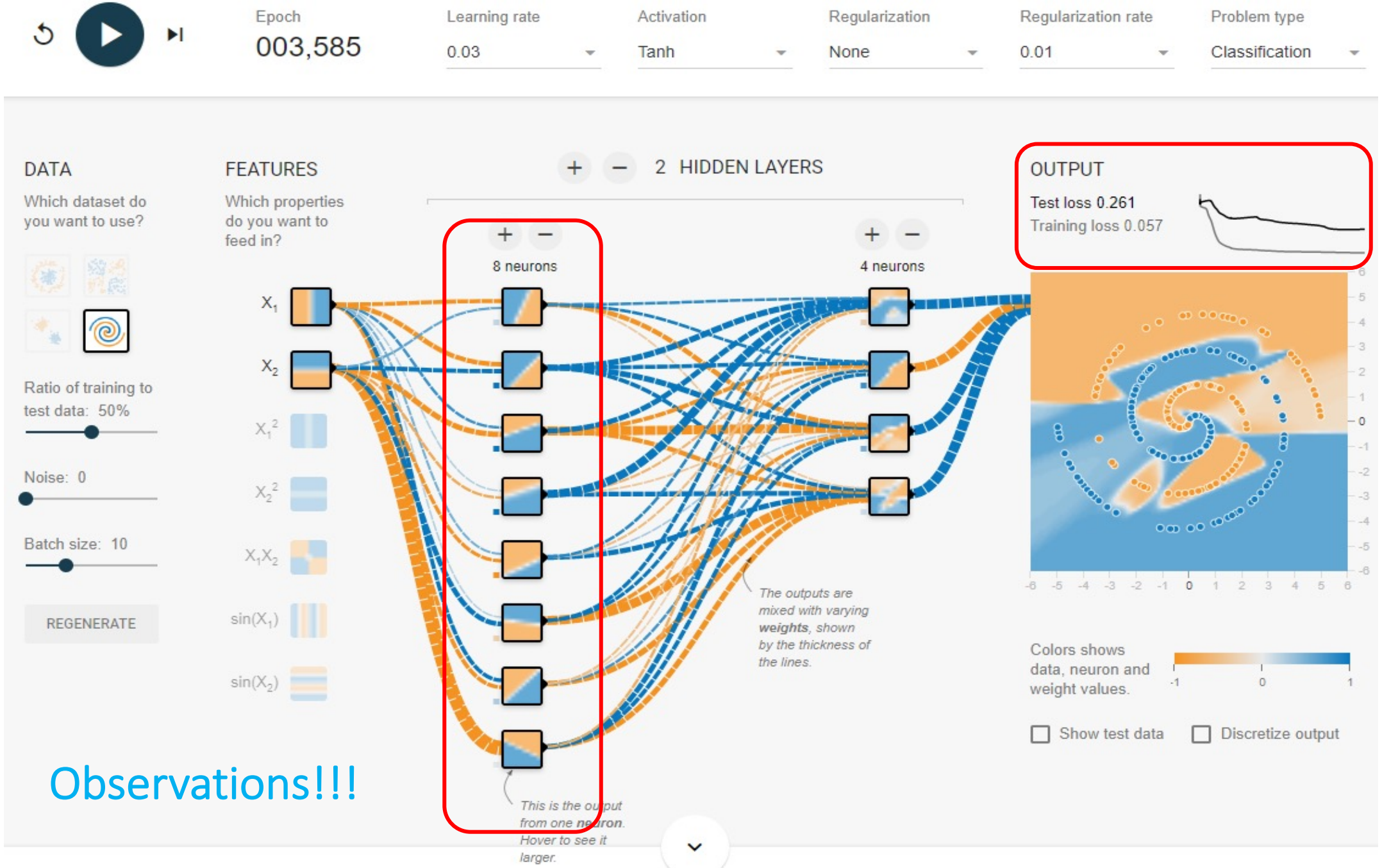
Observations!!!

Classification Experiment #6



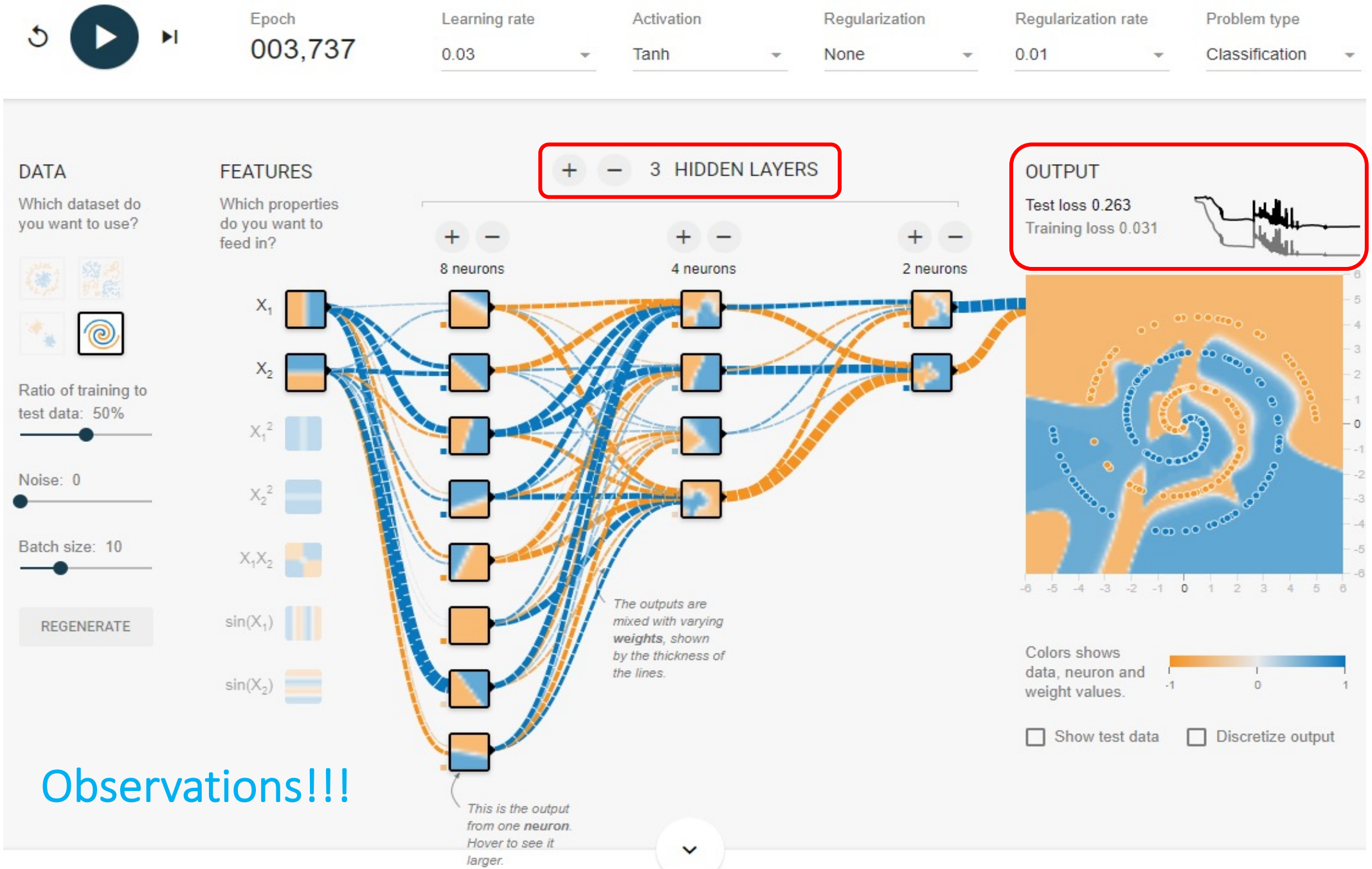
Observations!!!

Classification Experiment #7



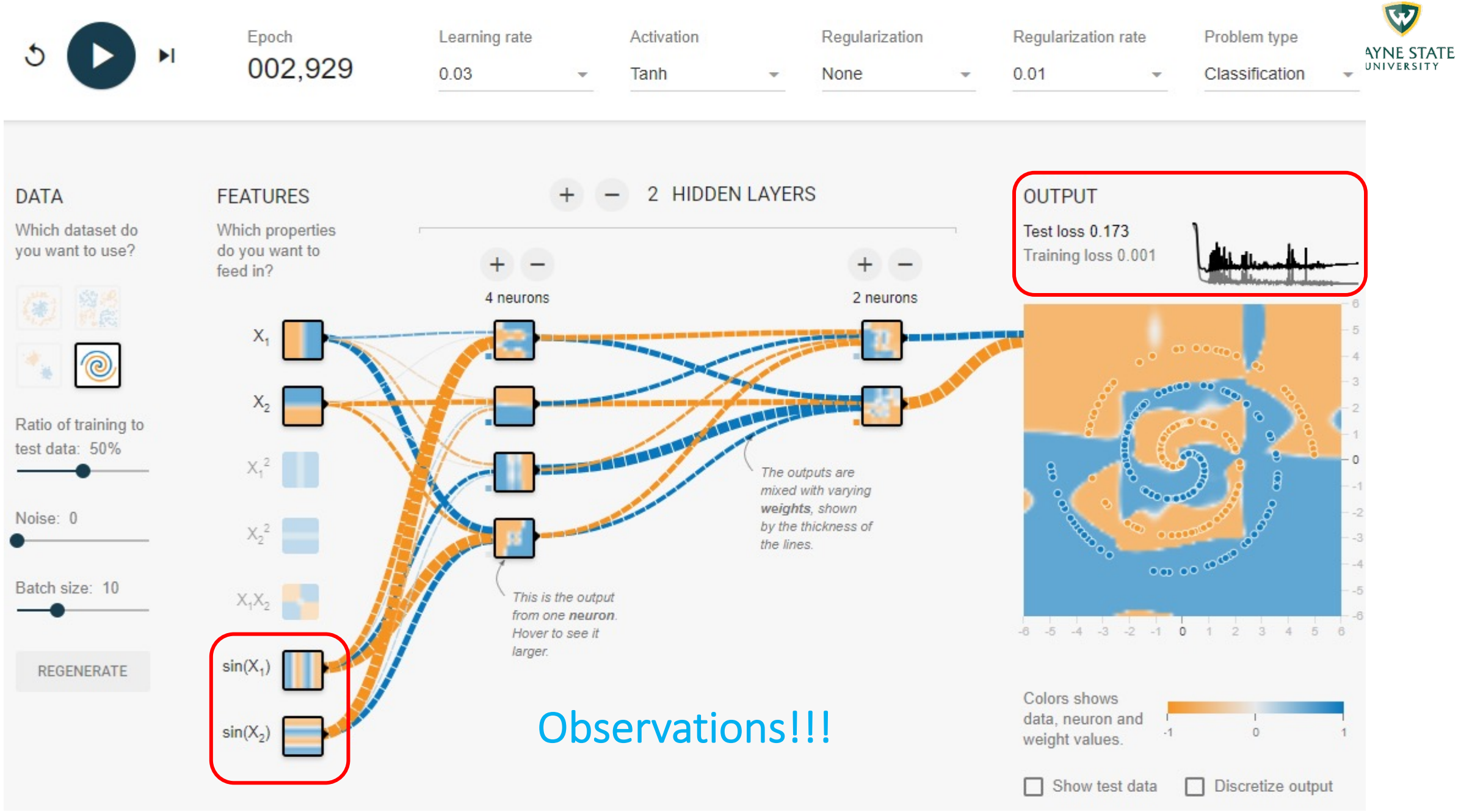
Observations!!!

Classification Experiment #8

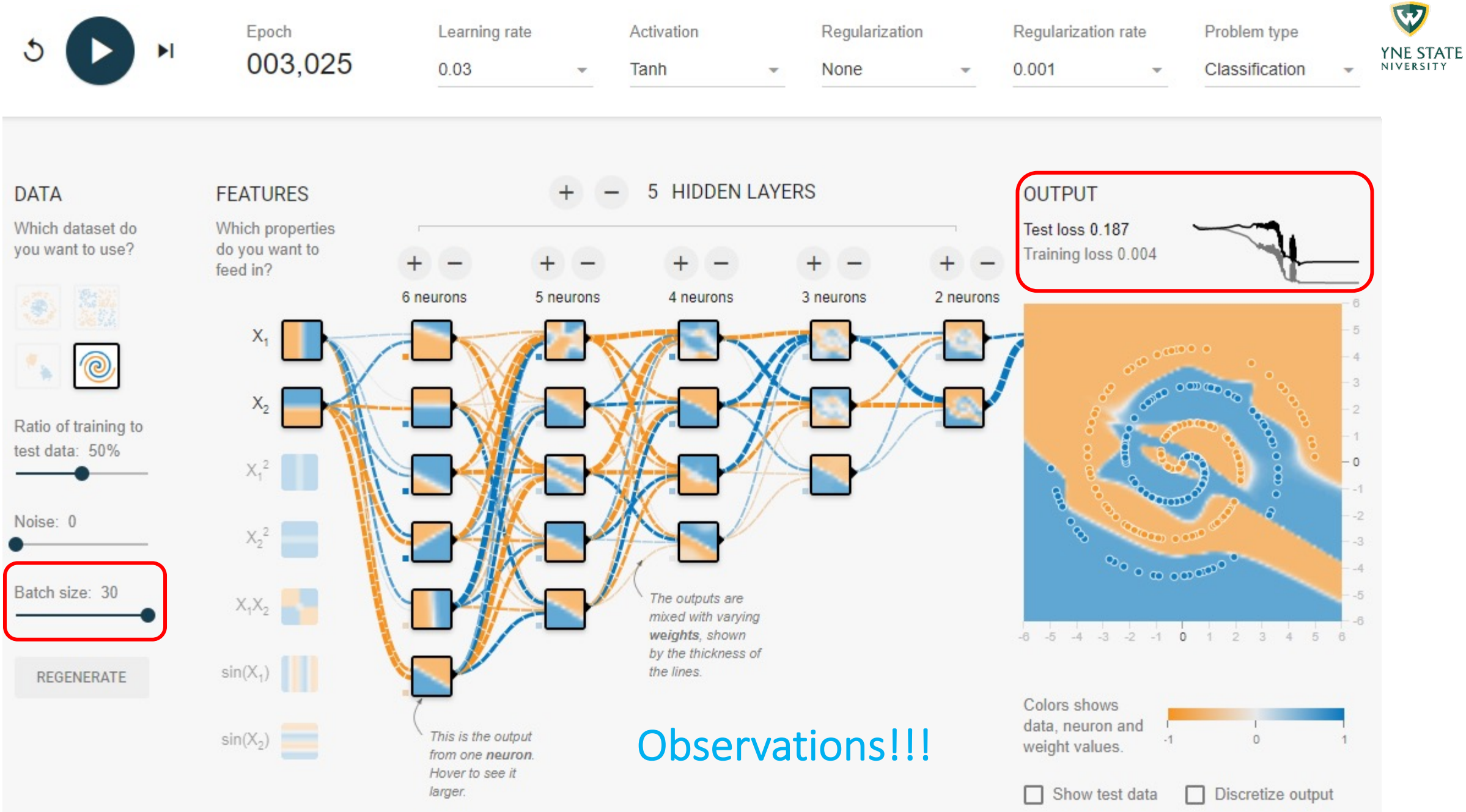


Observations!!!

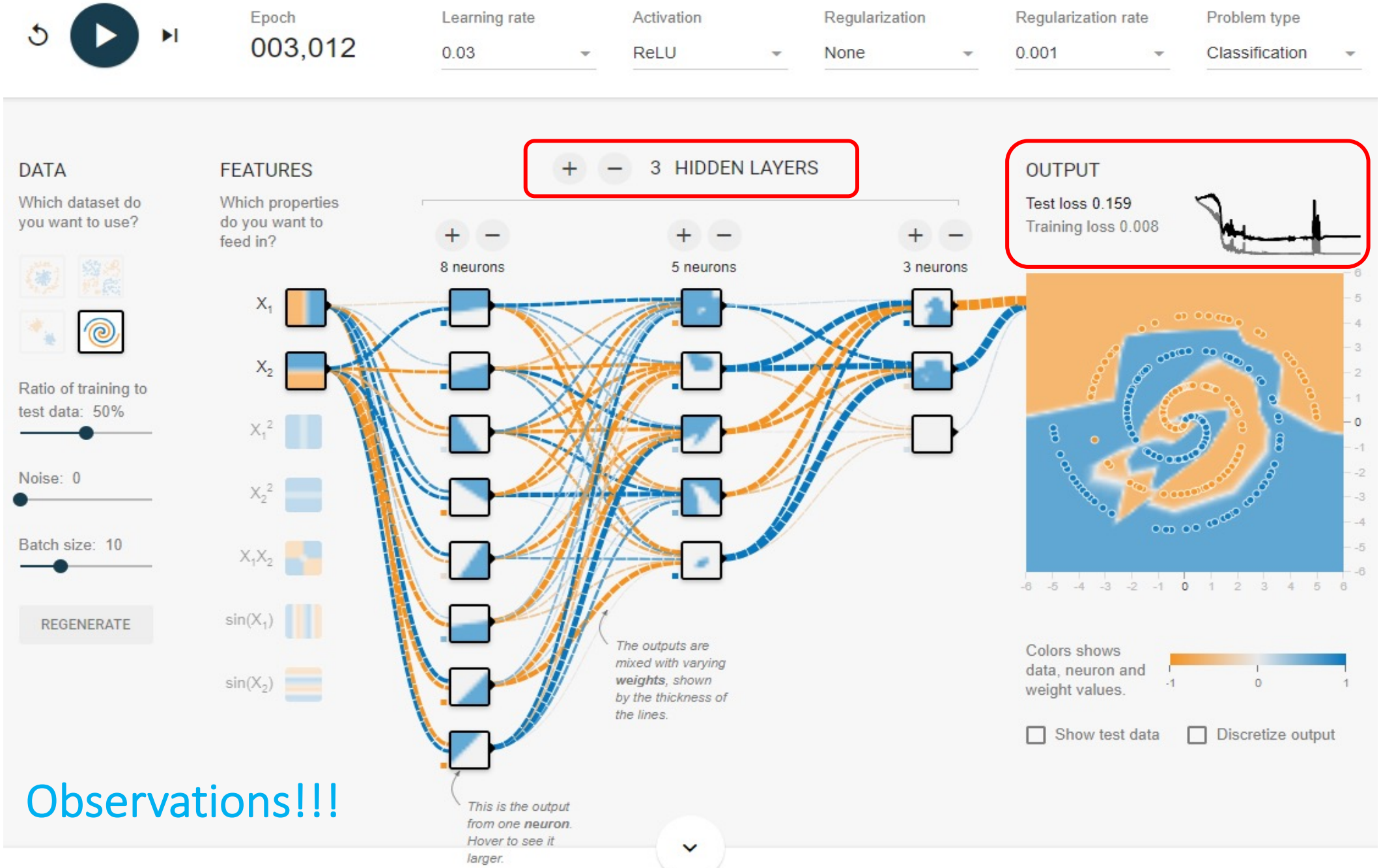
Classification Experiment #9



Classification Experiment #10



Classification Experiment #11



Observations!!!

Classification Experiment #12