SUPERVISED LEARNING AND PERCEPTRON

INT301 Bio-computation, Week 2, 2021

Outline

- McCulloch-Pitts neuron
- Hebb's learning rule
- Supervised learning model: Perceptron

Recall: Machine learning and ANN

- Like human learning from past experiences.
- A computer does not have “experiences”.
- A computer system learns from data, which represent some “past experiences” of an application domain.
- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., yes or no, and high or low.
- The task is commonly called: supervised learning.

The data and the goal

- **Data:** A set of data records (also called examples, instances or cases) described by
  - *k* attributes: A_1, A_2, ... A_k.
  - a class: Each example is labelled with a pre-defined class.
- **Goal:** To learn a classification model from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

Approve or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

An example: the learning task

- **Learn a classification model** from the data
- Use the model to classify future loan applications into
  - **Yes (approve**) and
  - **No (disapprove)**
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

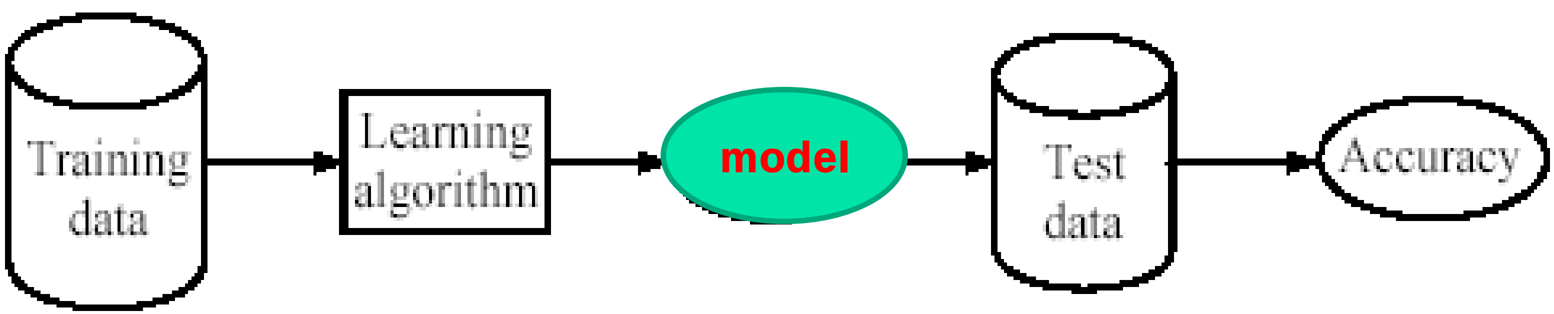
Supervised vs. unsupervised Learning

- **Supervised learning**: classification is seen as supervised learning from examples.
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a “teacher” gives the classes (supervision).
  - Test data are classified into these classes too.
- **Unsupervised learning** (e.g. clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- Learning (training): Learn a model using the training data
- Testing: Test the model using **unseen** test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



Step 1. Training **Step 2. Testing**

What do we mean by learning?

- Given
  - a data set *D*
  - a task *T*
  - a performance measure *M*

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as compared to no learning.

Fundamental assumption of learning

Assumption: The distribution of training examples is **identical** to the distribution of test examples (including future unseen examples).

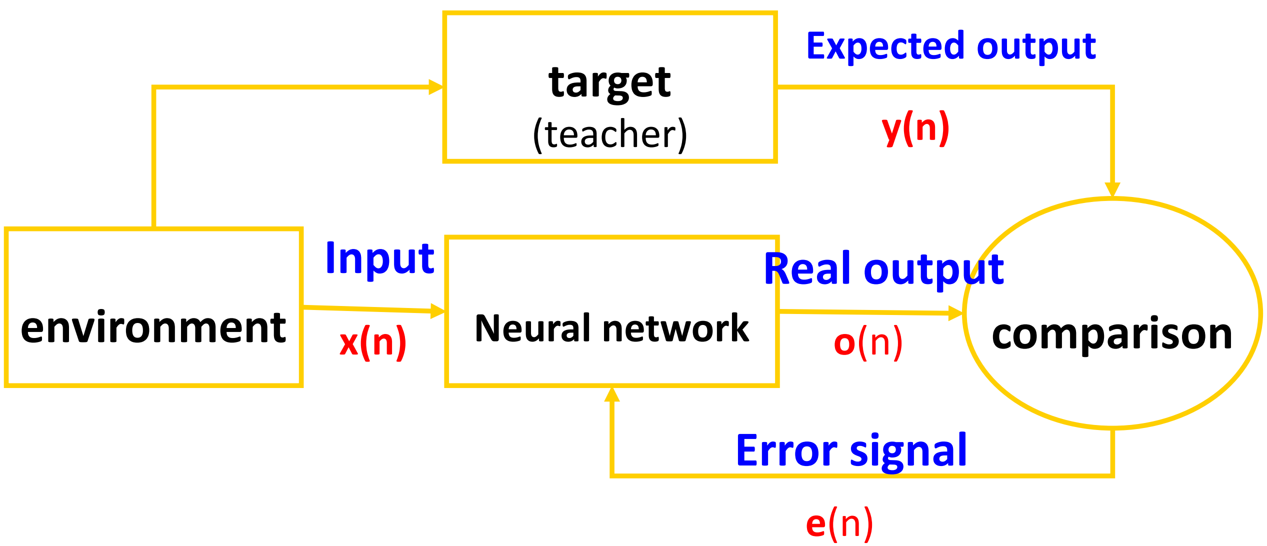
- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Perceptron (1958)

- Rosenblatt (1958) explicitly considered the problem of **pattern recognition**, where a “teacher” is essential.
- Perceptrons are neural networks that change with “experience” using ***error-correcting rule***.
- According to the rule, weight of a response unit changes when it makes erroneous response to stimuli presented to the network.

ANN for Pattern Recognition

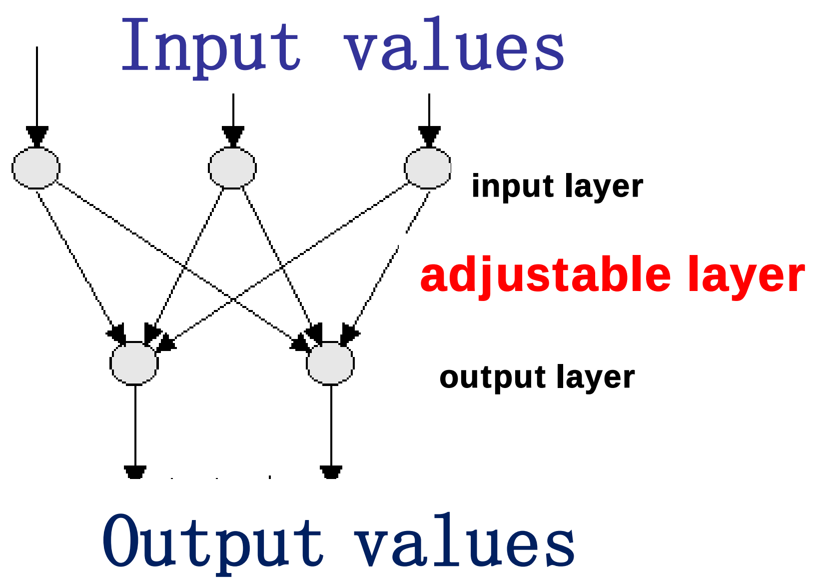
- **Training data**: set of sample pairs (x, y).
- Network (model, classifier) **adjusts its connection weights according to the errors** between target and network output



Supervised learning is mainly applied in classification/prediction.

Perceptron (1958)

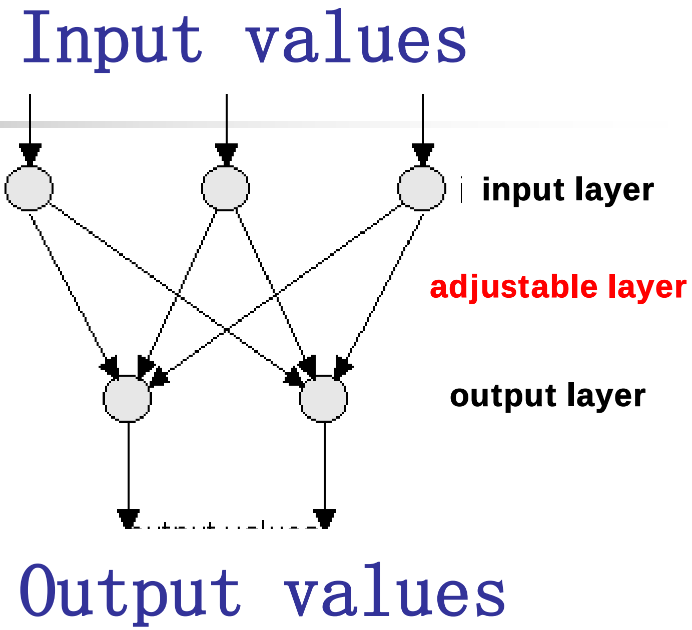
- The simplest architecture of perceptron comprises two layers of idealised “neurons”, which we shall call *“units” of the network*.



- There are
  - one layer of input units, and
  - one layer of output units.

in the perceptron

Perceptron (1958)

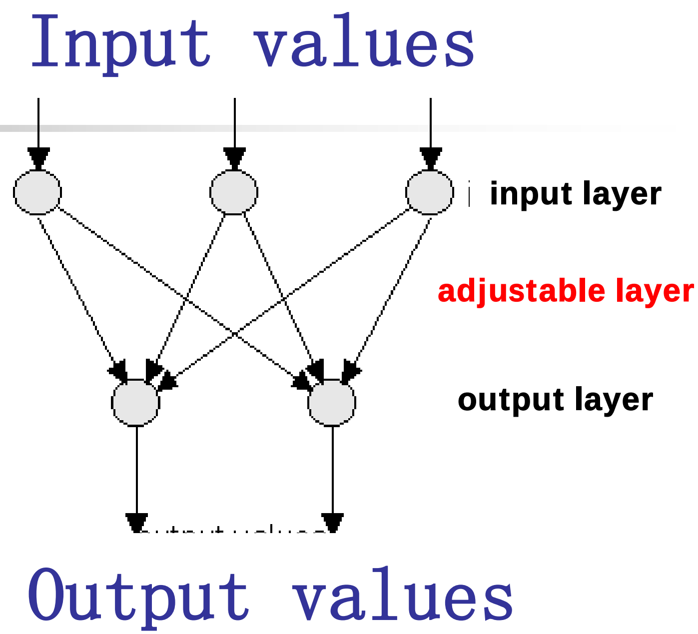


- The two layers are fully interconnected, *i.e.,* every input unit is connected to every output unit
- Thus, *processing elements* of the perceptron are the ***abstract neurons***
- Each processing element has the same input comprising total input layer, but individual outputs with individual connections and therefore different weights of connections.

Perceptron (1958)

The total input to the output unit j is

$$S_j = \sum_{i=0}^{n} w_{ji} a_i$$



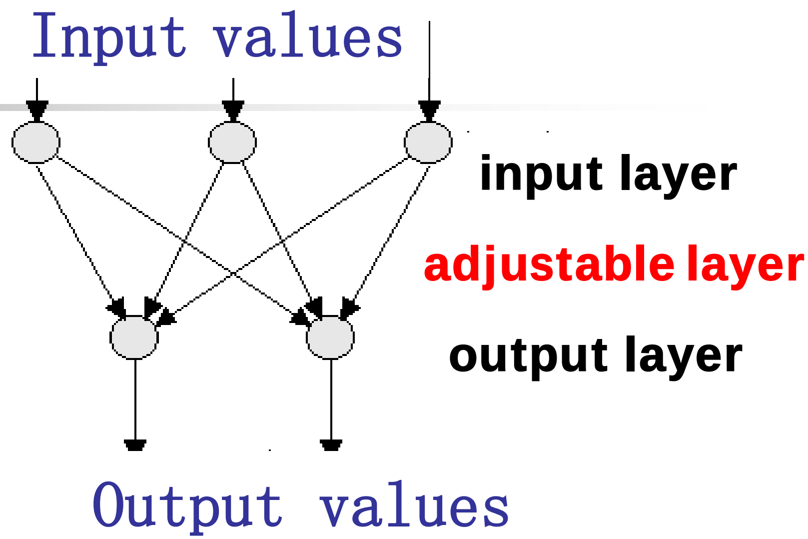
a_i : input value from the ith input unit

w_{ji} : the weight of connection btw i-th input and j-th output units

- The sum is taken over all n+1 inputs units connected to the output unit j.
- There is special bias input unit *number 0* in the input layer.

Perceptron (1958)

$$S_j = \sum_{i=0}^{n} w_{ji} a_i$$



i=0

- There is a special **bias input** unit **number 0** in the input layer.
- The bias unit always produces inputs a_0 of the fixed values of +1.
- The input a_0 of **bias unit** functions as a constant value in the sum.
- The **bias unit connection** to output unit j has a weight w_{j0} adjusted in the same way as all the other weights

Perceptron (1958)

- The output value X_j of the output unit j depends on whether the weighted sum is above or below the unit's threshold value.
- X_j is defined by the unit's threshold activation function.

$$X_j = f(S_j) = \begin{cases} 1, S_j \geq \theta_j \\ 0, S_j < \theta_j \end{cases}$$

Definition:

the ordered set of instant outputs of all units in the output layer $X = \{X_0, X_1, ..., \mathrm{X_n}\}$

constitutes an **output vector** of the network

Perceptron (1958)

- The instant output X_j of the j-th unit in the output layer constitutes the j-th component of the output vector.
- Weight w_{ji} of connections between the two layers are changed according to **perceptron learning rule**, so the network is more likely to produce the desired output in response to certain inputs.
- The process of weights adjustment is called **perceptron learning (or training).**

Perceptron Training

Every processing element computes an output according its state and threshold:

$$S_j = \sum_{i=0}^{n} w_{ji} a_i$$

$$X_j = f(S_j) = \begin{cases} 1, S_j \geq \theta_j \\ 0, S_j < \theta_j \end{cases}$$

The network instant outputs X_j are then compared to the desired outputs specified in the training set.

$$\boxed{e_j = (t_j - X_j)}$$

The error of an output unit is the difference between the target output and the instant one.

Perceptron Training

$$S_j = \sum_{i=0}^{n} w_{ji} a_i$$

The error are computed and

used to re-adjust the values

of the weights of connections.

$$X_j = f(S_j) = \begin{cases} 1, S_j \geq \theta_j \\ 0, S_j < \theta_j \end{cases}$$

$$\boxed{e_j = (t_j - X_j)}$$

The weights re-adjustment is done in such a way that the network is – on the whole – more likely to give the desired response next time.

Perceptron Updating of the Weights

The goal of the training session is to arrive at a single set of weights that allow each of the mappings in the training set to be done successfully by the network.

1. Compute *error* of every output unit

$$\boxed{e_j = (t_j - X_j)}$$

where

t_j is the target value for output unit j

X_j is the instant output produced by output unit j

Perceptron Updating of the Weights

Having the errors computed,

2. **Update the weights**

$$\boxed{\underset{\text{new}}{w_{ji}} = \underset{\text{old}}{w_{ji}} + \Delta w_{ji}}$$

where

$$\boxed{\Delta w_{ji} = Ce_j a_i = C(t_j - X_j)a_i}$$

Perceptron learning rule

Perceptron training

- A sequential learning procedure for updating the weights.
- Perceptron training algorithm (delta rule)

$$\boxed{\Delta w = \text{learning rate} \times \underbrace{(\text{teacher} - \text{output})}_{\text{error}} \times \text{input}}$$

Example

- Define our "features":

Taste	Sweet = 1, Not_Sweet = 0
Seeds	Edible = 1, Not_Edible = 0
Skin	Edible = 1, Not_Edible = 0

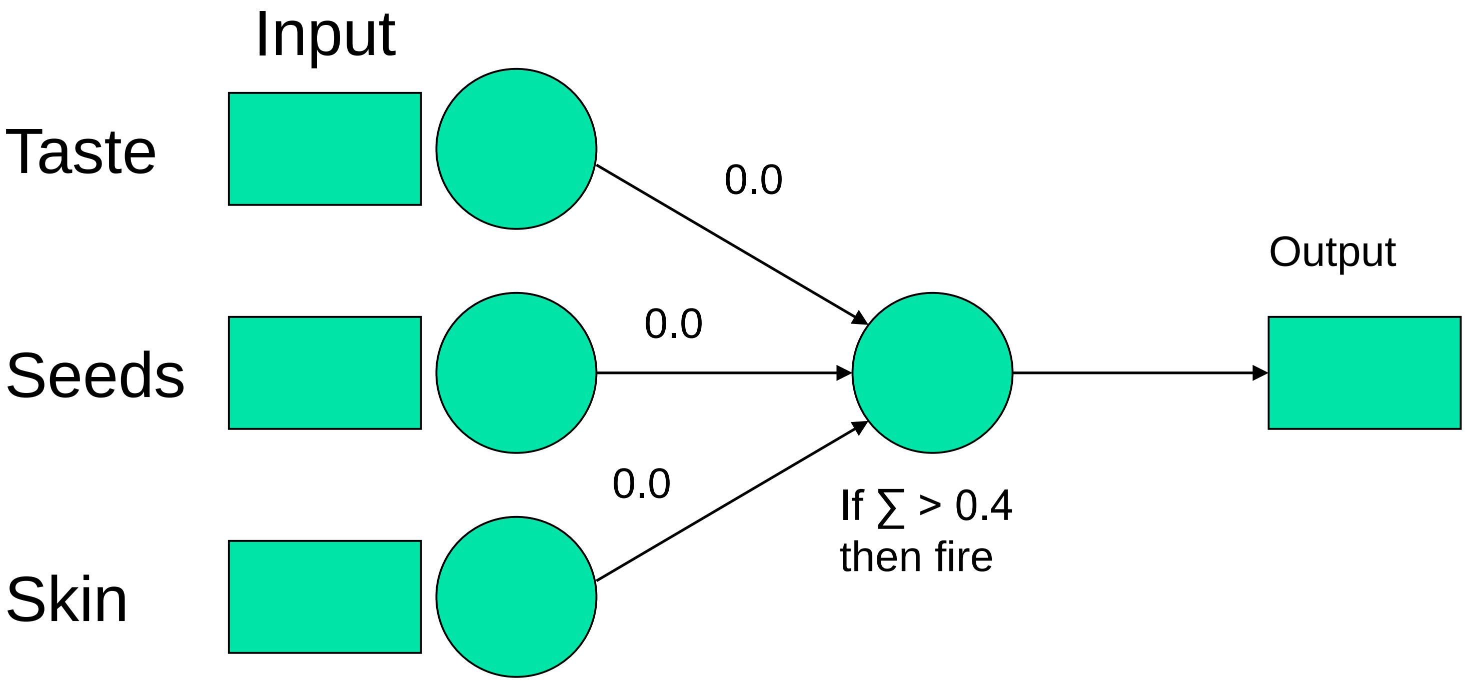
For output:

Good_Fruit = 1

Not_Good_Fruit = 0

Example

Let's start with no knowledge:



Example

- To train the perceptron, we will show it each example and have it categorize each one.
- Since it's starting with no knowledge, it is going to make mistakes. When it makes a mistake, we are going to adjust the weights to make that mistake less likely in the future.

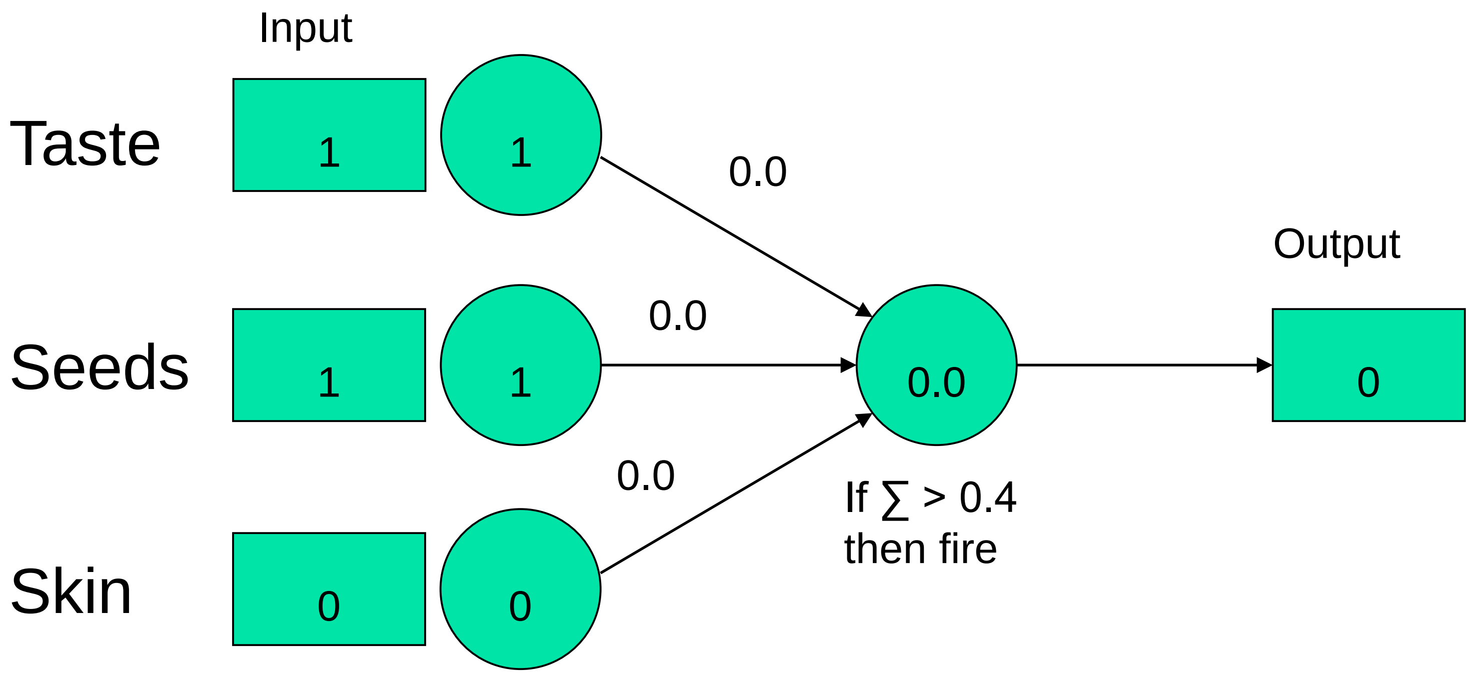
Example

- When we adjust the weights, we’re going to take relatively small steps to be sure we don’t over-correct and create new problems.

- We’re going to learn the category “good fruit” defined as anything that is sweet.
  - **`Good fruit = 1`**
  - **`Not good fruit = 0`**

Example

Show it a banana:



Example

- In this case we have:

  (1 X 0) = 0

  + (1 X 0) = 0

  + (0 X 0) = 0

- It adds up to 0.0.
- Since that is less than the threshold (0.40), we responded “no.”
- Is that correct? No.

Example

- Since we got it wrong, we need to change the weights. We'll do that using the **delta rule** (delta for change).

$$\boxed{\Delta w = \text{learning rate} \times (\text{teacher} - \text{output}) \times \text{input}}$$

Example

The three parts of that are:

- Learning rate: We set that ourselves. Set large enough that learning happens in a reasonable amount of time; and also small enough to avoid too fast. Here pick 0.25.
- (teacher - output): The teacher knows the correct answer (e.g., that a banana should be a good fruit). In this case, the teacher says 1, the output is 0, so (1 - 0) = 1.
- Input: That's what came out of the node whose weight we're adjusting. For the first node, 1.

Example

- To pull it together:
  - Learning rate: 0.25.
  - (teacher - output): 1.
  - input: 1.

$$\Delta w = 0.25 \times 1 \times 1 = 0.25.$$

- Since it's a Δw, it's telling us how much to change the first weight. In this case, we're adding 0.25 to it.

Example

Let's think about the delta rule:

(teacher - output)

- If we get the categorization right, (teacher - output) will be zero (the right answer minus itself).
- In other words, if we get it right, we won't change any of the weights. As far as we know we have a good solution, why would we change it?

Example

Let's think about the delta rule:

(teacher - output)

- If we get the categorization wrong, (teacher - output) will either be -1 or +1.
  - If we said "yes" when the answer was "no", we're too high on the weights and we will get a (teacher - output) of -1 which will result in reducing the weights.
  - If we said "no" when the answer was "yes", we're too low on the weights and this will cause them to be increased.

Example

Let's think about the delta rule:

- Input:
  - If the node whose weight we're adjusting sent in a 0, then it didn't participate in making the decision. In that case, it shouldn't be adjusted. Multiplying by zero will make that happen.
  - If the node whose weight we're adjusting sent in a 1, then it did participate and we should change the weight (up or down as needed) if the corresponding output wrong.

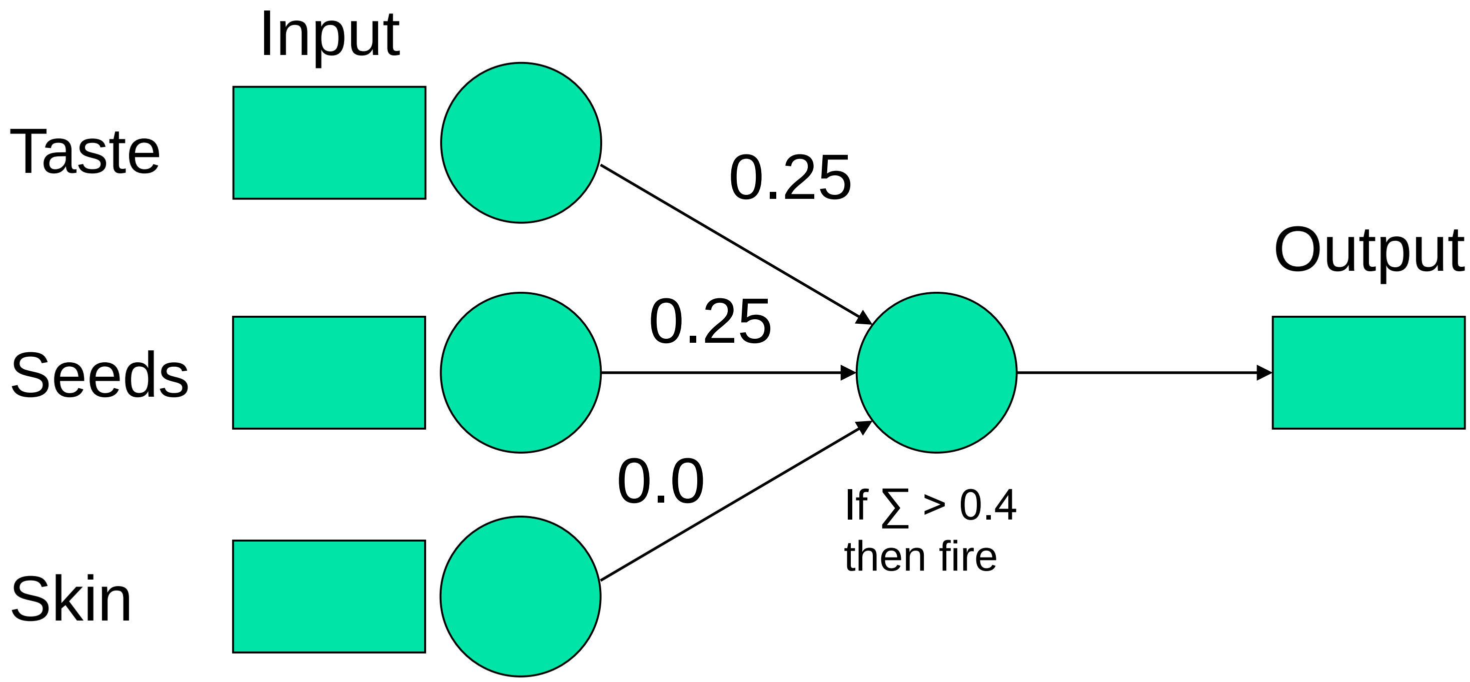
Example

- How do we change the weights for banana?

Feature:	Learning rate:	(teacher - output):	Input:	Δw
taste	0.25	1	1	+0.25
seeds	0.25	1	1	+0.25
skin	0.25	1	0	0

Example

Here it is with the adjusted weights:



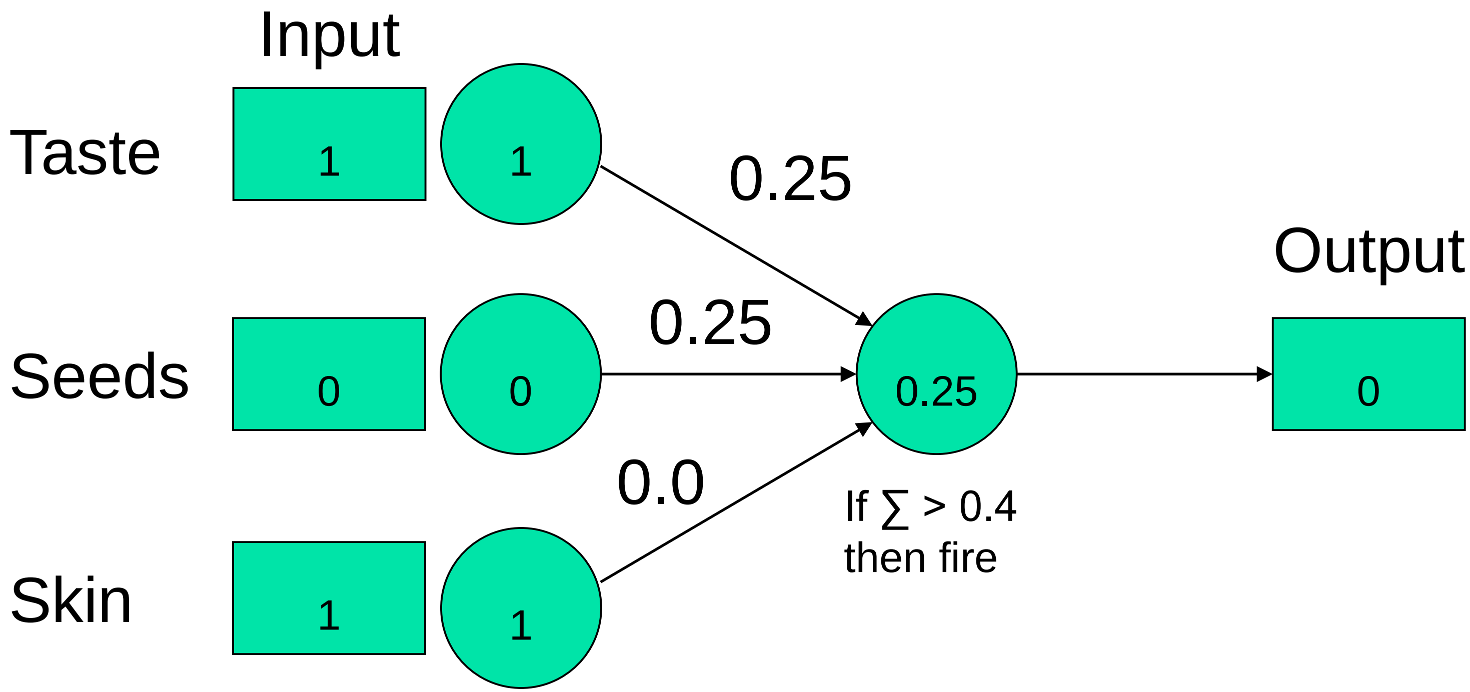
Example

- To continue training, we show it the next example, adjust the weights…

- We will keep cycling through the examples until we go all the way through one time without making any changes to the weights. At that point, the concept is learned.

Example

Show it a pear:



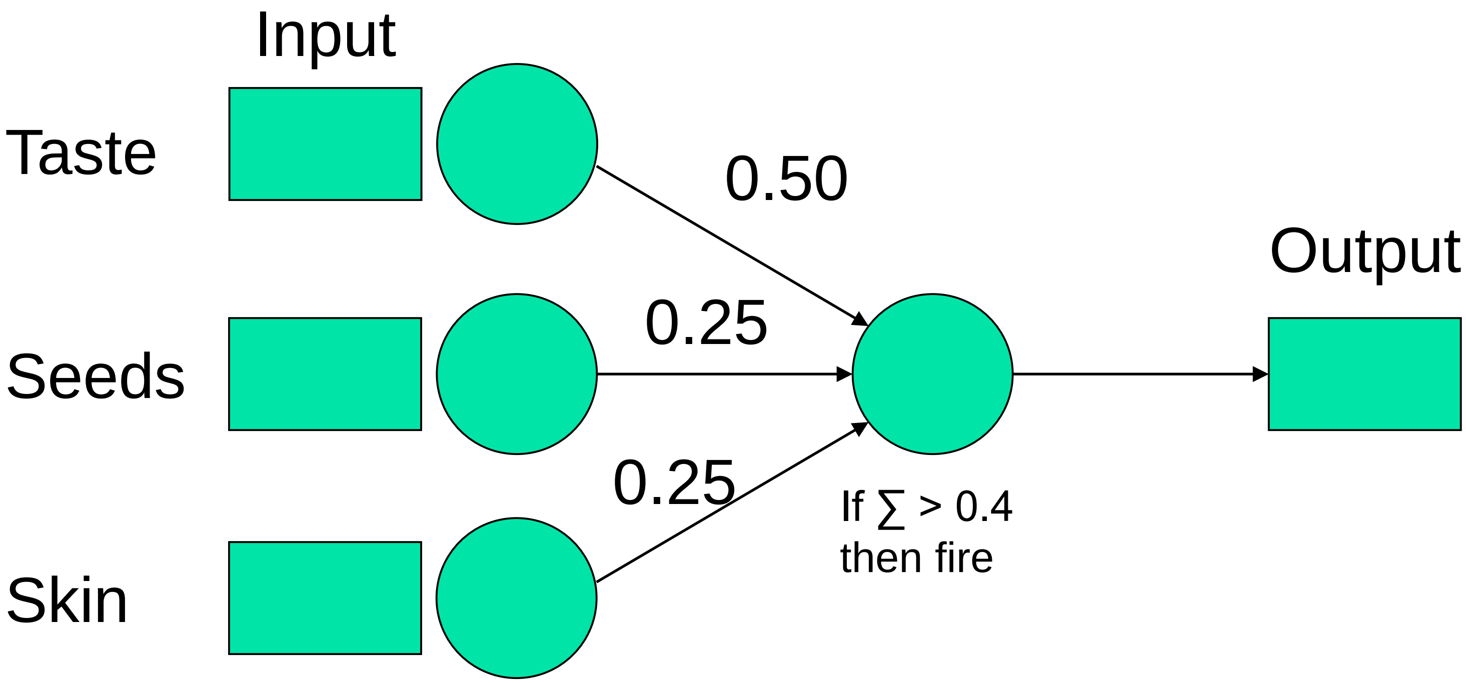
Example

- How do we change the weights for pear?

Feature:	Learning rate:	(teacher - output):	Input:	Δw
taste	0.25	1	1	+0.25
seeds	0.25	1	0	0
skin	0.25	1	1	+0.25

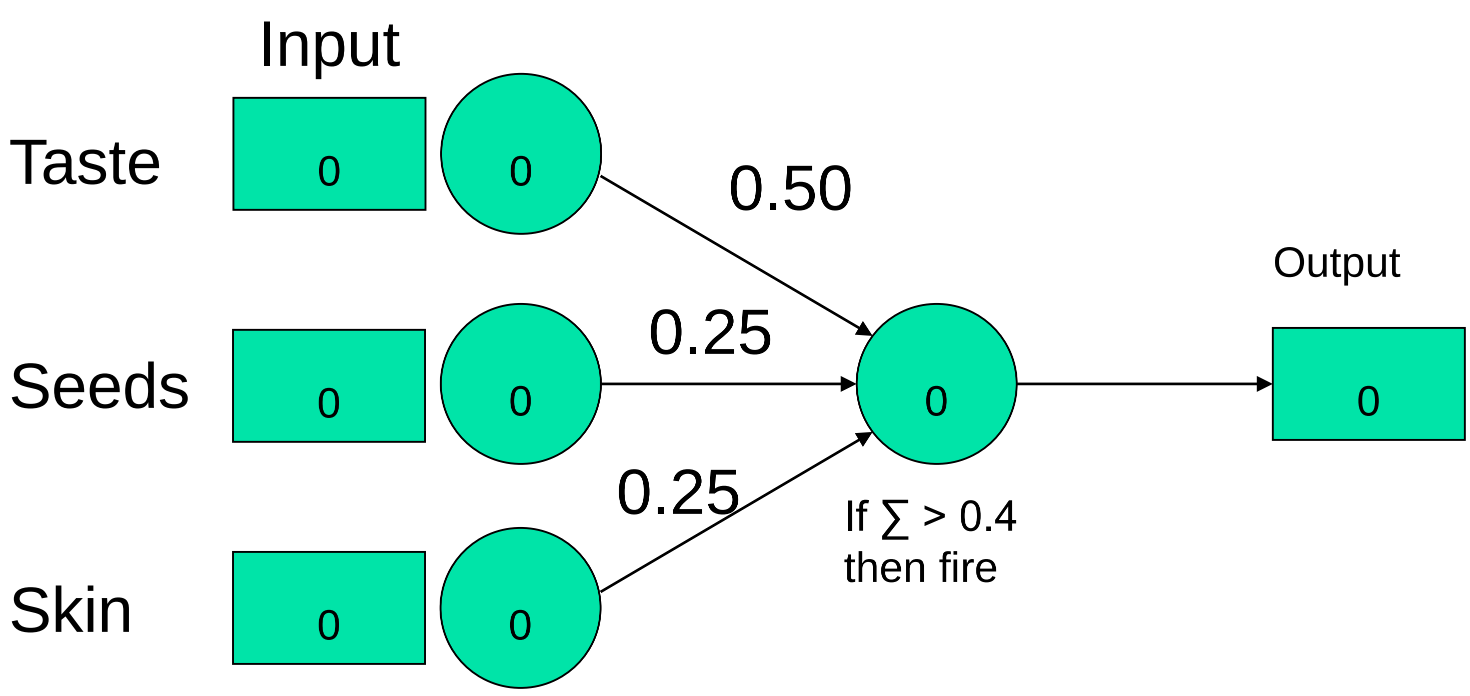
Example

Here it is with the adjusted weights:



Example

Show it a lemon:



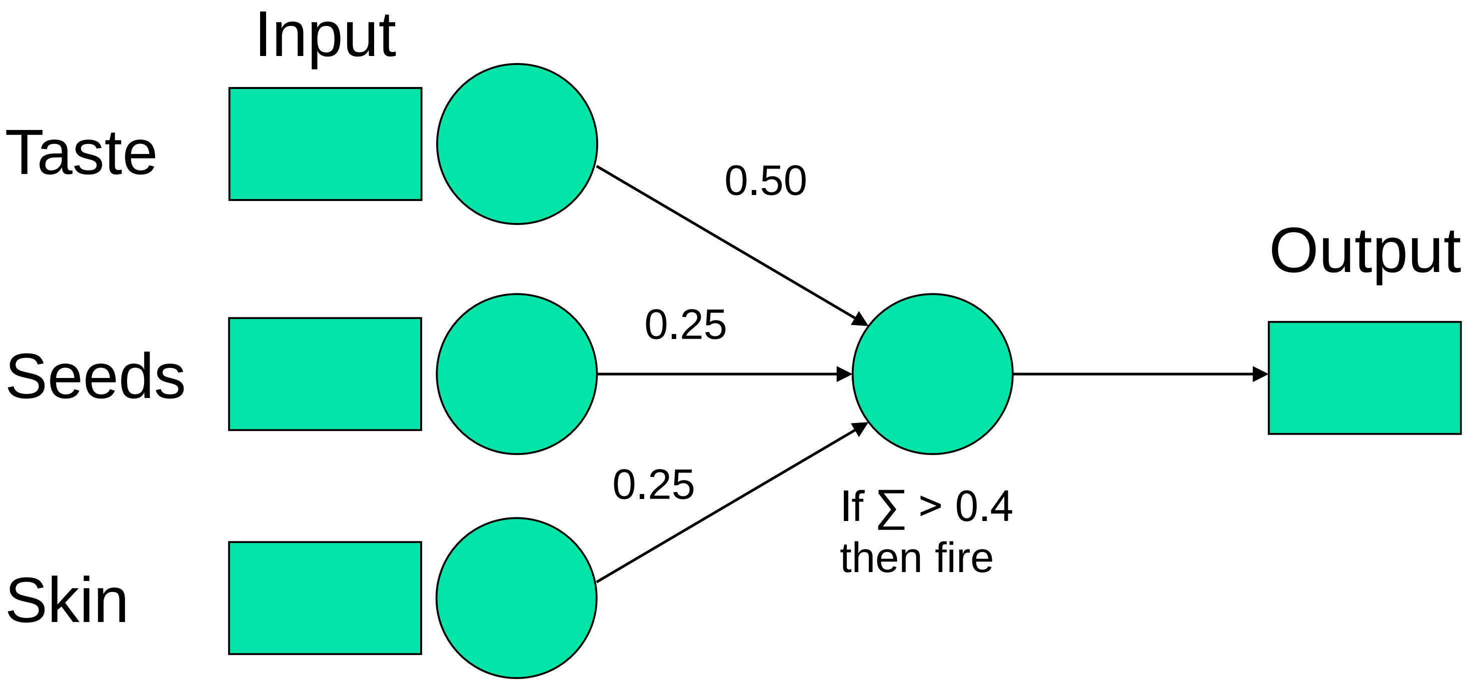
Example

- How do we change the weights for lemon?

Feature:	Learning rate:	(teacher - output):	Input:	Δw
taste	0.25	0	0	0
seeds	0.25	0	0	0
skin	0.25	0	0	0

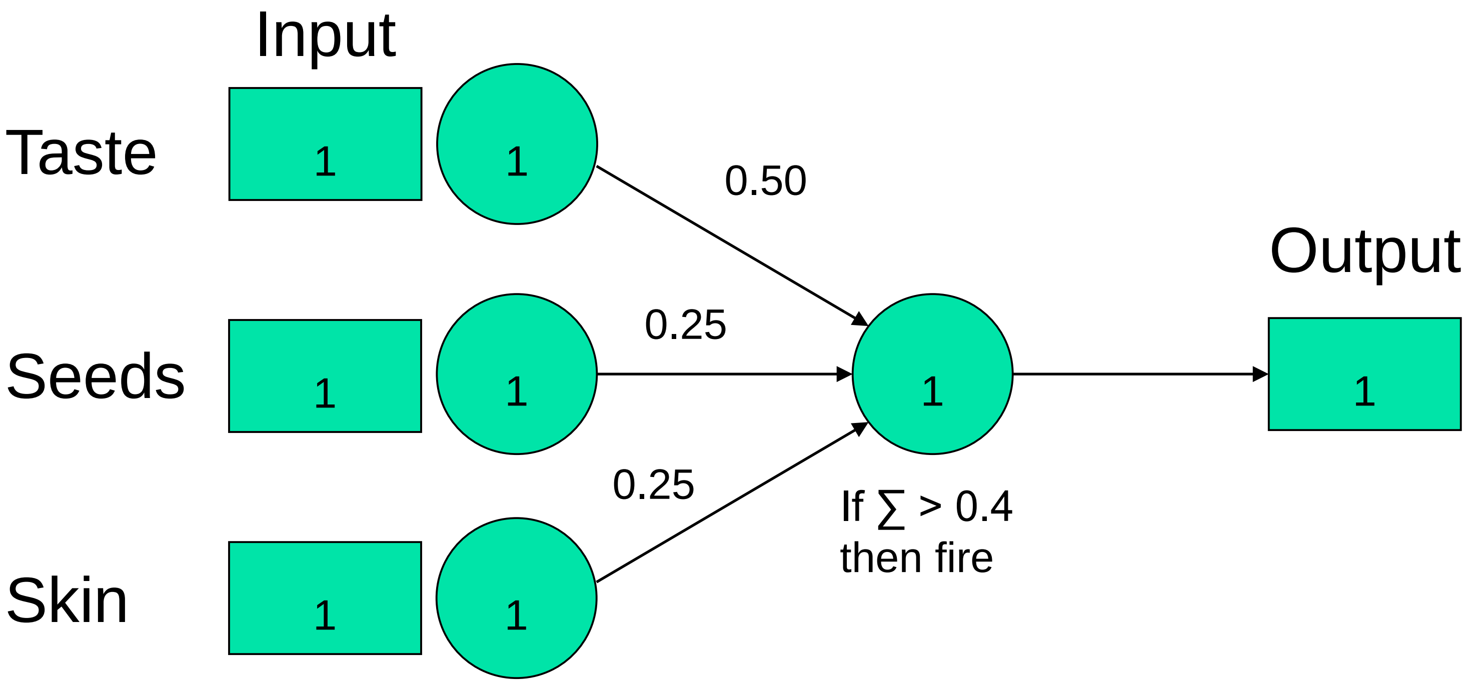
Example

Here it is with the adjusted weights:



Example

Show it a strawberry:



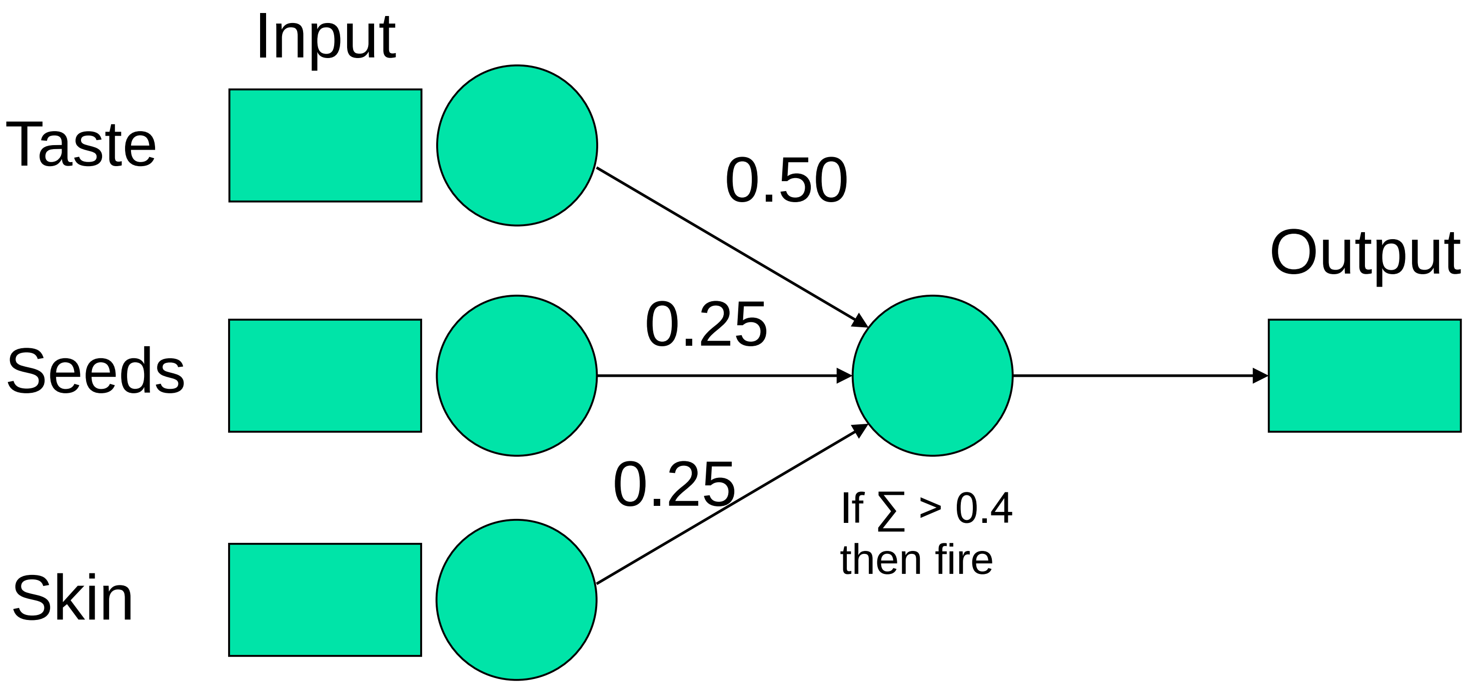
Example

- How do we change the weights for strawberry?

Feature:	Learning rate:	(teacher - output):	Input:	Δw
taste	0.25	0	1	0
seeds	0.25	0	1	0
skin	0.25	0	1	0

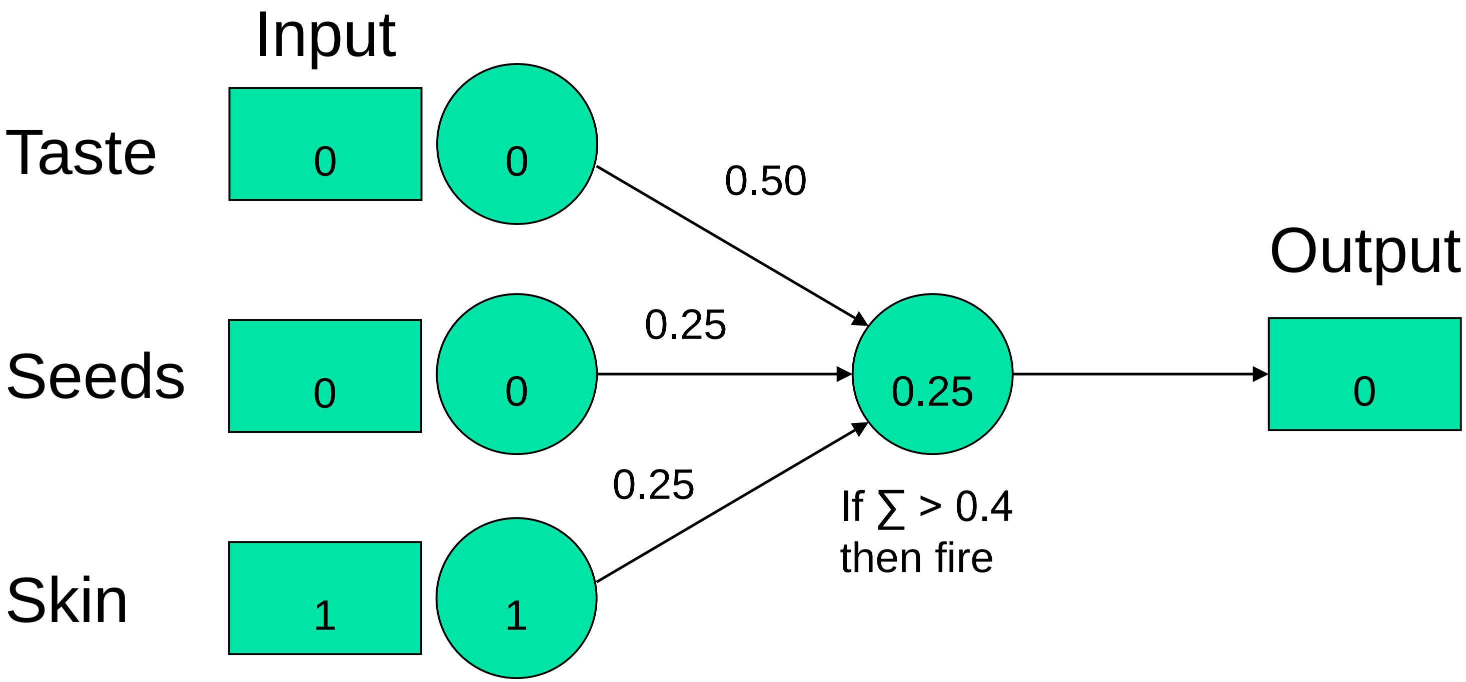
Example

Here it is with the adjusted weights:



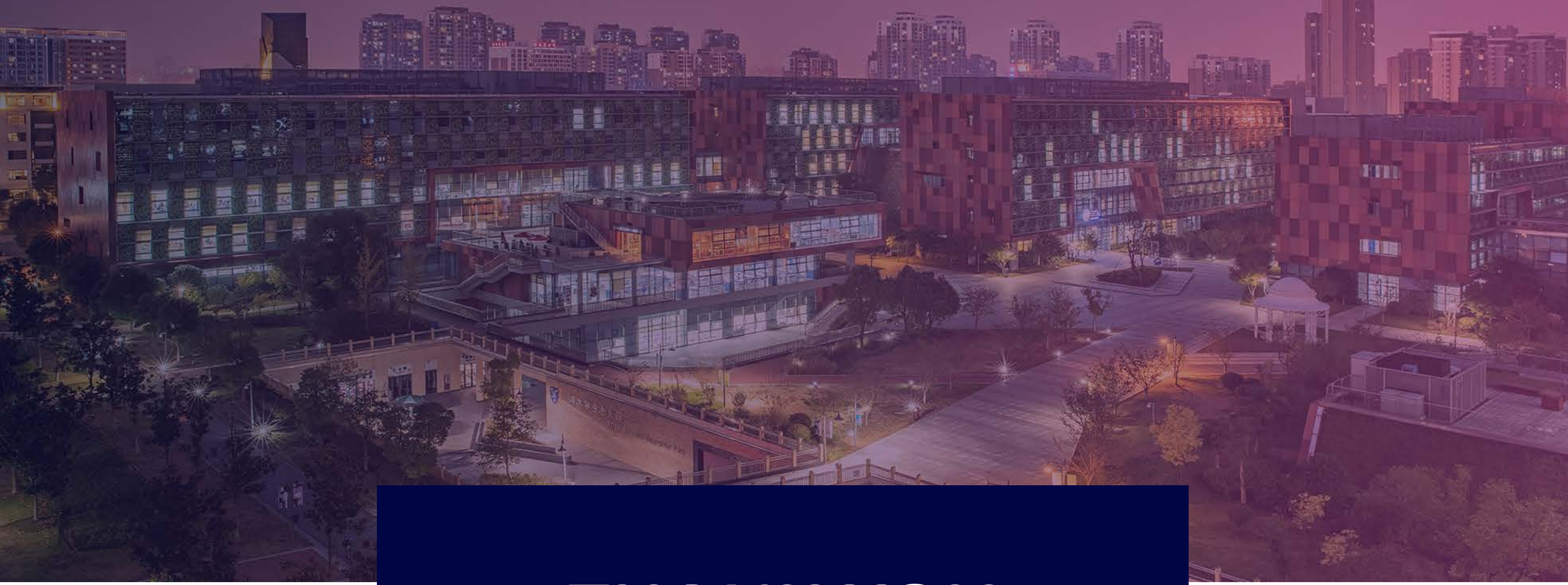
Example

Show it a green apple:



Example

- If you keep going, you will see that this perceptron can correctly classify the examples that we have.



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi’an Jiaotong-Liverpool University
西交利物浦大学