Tutorial : SOM
Self Organizing Map

SYD 522

Sobhan Hemati

University of Waterloo

Today’s Agenda

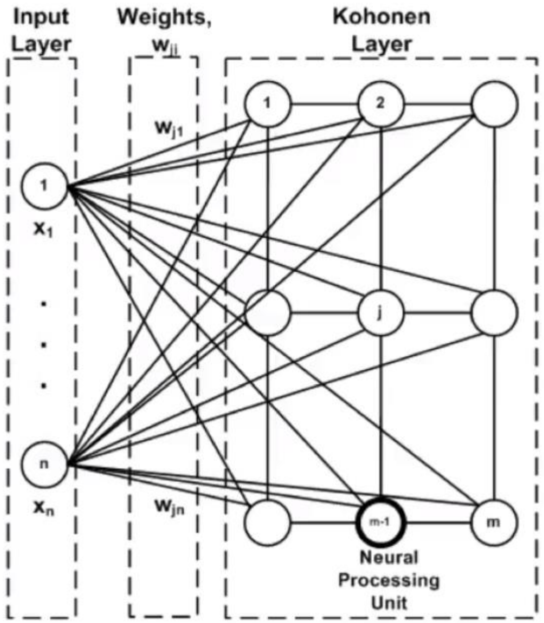
- SOM in a qualitative example
- SOM in a quantitative example
- Dimension reduction using SOM
- SOM in notebook

What is SOM?

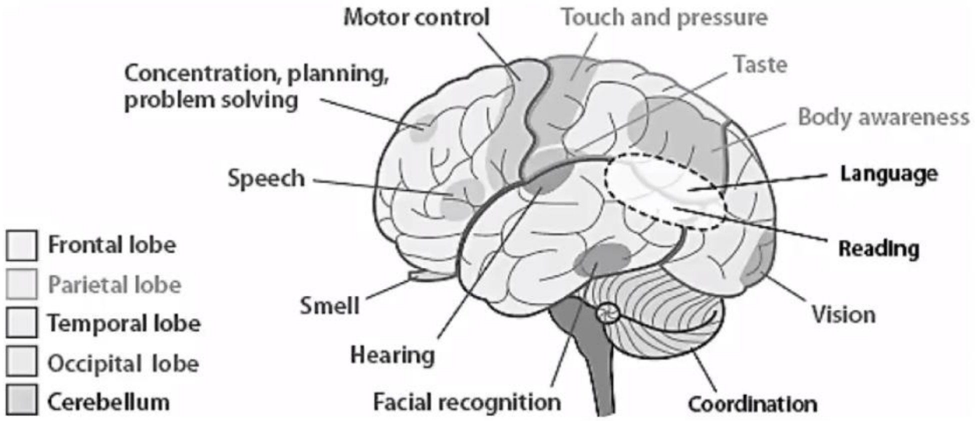
A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction….they apply competitive learning as opposed to error-correction learning….they use a neighborhood function to preserve the topological properties of the input space.

Ref: Wikipedia

- Self-organizing maps are unsupervised neural networks that cluster high-dimensional data
- Transform complex inputs into easy to understand two-dimensional outputs

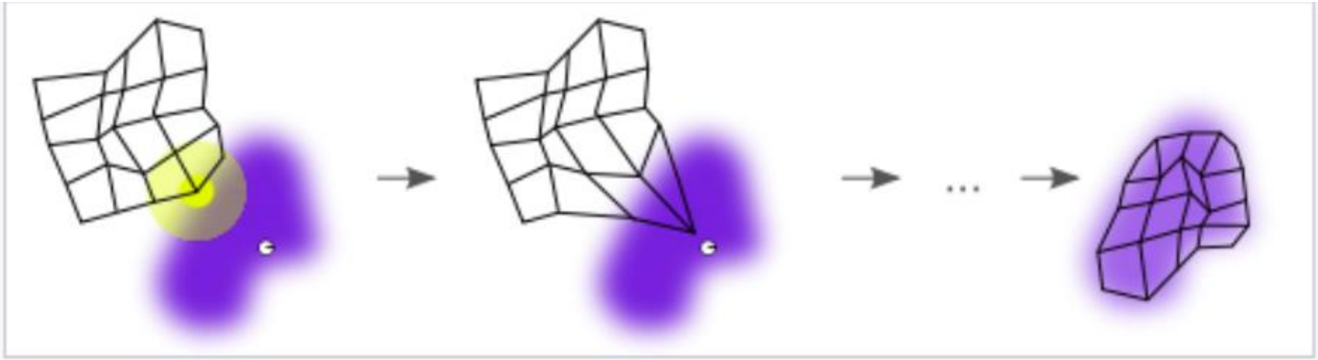


Araujo, Ernesto & R. Silva, Cassiano & J. B. S. Sampaio, Daniel. (2008). Video Target Tracking by using Competitive Neural Networks. WSEAS Transactions on Signal Processing. 4.



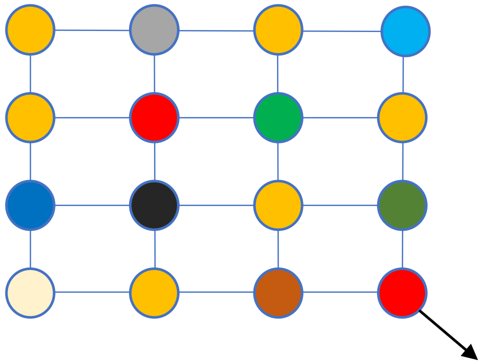
http://mlexplore.org/2017/01/13/self-organizing-maps-in-go/

SOM Algorithm



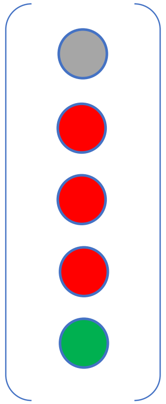
1. **def** som(data):
2. create a 2D lattice
3. **for** d_i in **data**:
4. w = find **winning neuron** in the lattice
5. update the weights of v^{th} neuron:
6. $w_v = w_v + \theta_t \alpha_t (d_i - w_v)$
7. **goto** 3. if **not converged**

SOM Algorithm: STEP 1



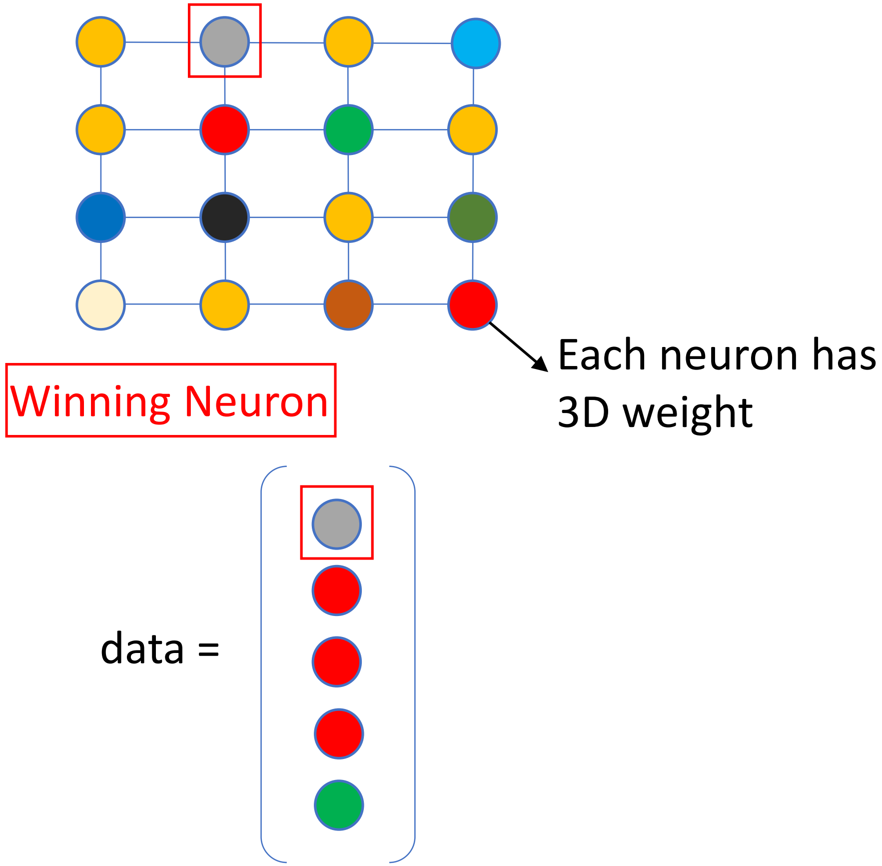
Each neuron has 3D weight

data =



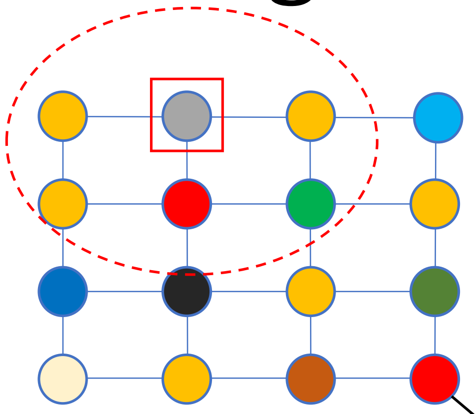
1. **def som(data):**
2.   **create a 2D lattice**
3.   **for** d_i in **data:**
4.   w = find **winning neuron** in the lattice
5.   update the weights of v^{th} neuron:
6. $w_v = w_v + \theta_t \alpha_t (d_i - w_v)$
7.   **goto** 3. if **not converged**

SOM Algorithm: STEP 2



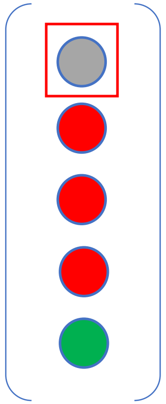
1. **def** som(data):
2. create a 2D lattice
3. **for d_i in data:**
4. **w = find winning neuron in the lattice**
5. update the weights of v^{th} neuron:
6. $w_v = w_v + \theta_t \alpha_t (d_i - w_v)$
7. **goto** 3. if **not converged**

SOM Algorithm: STEP 3



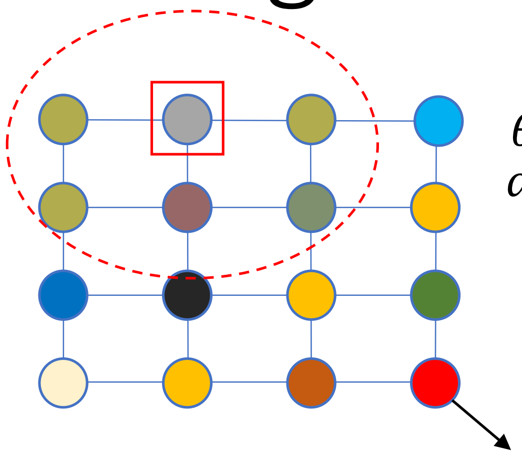
θ_t: Neighborhood function
α_t: Learning Rate

Each neuron has 3D weight

data = 

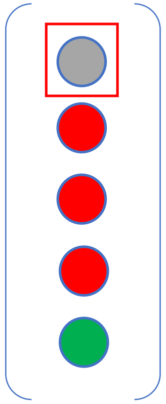
1. **def** som(data):
2.   create a 2D lattice
3.   **for** d_i in **data**:
4.   w = find **winning neuron** in the lattice
5.   **update the weights of v^{th} neuron:**
6. $\boldsymbol{w_v = w_v + \theta_t \alpha_t (d_i - w_v)}$
7.   **goto** 3. if **not converged**

SOM Algorithm: STEP 4



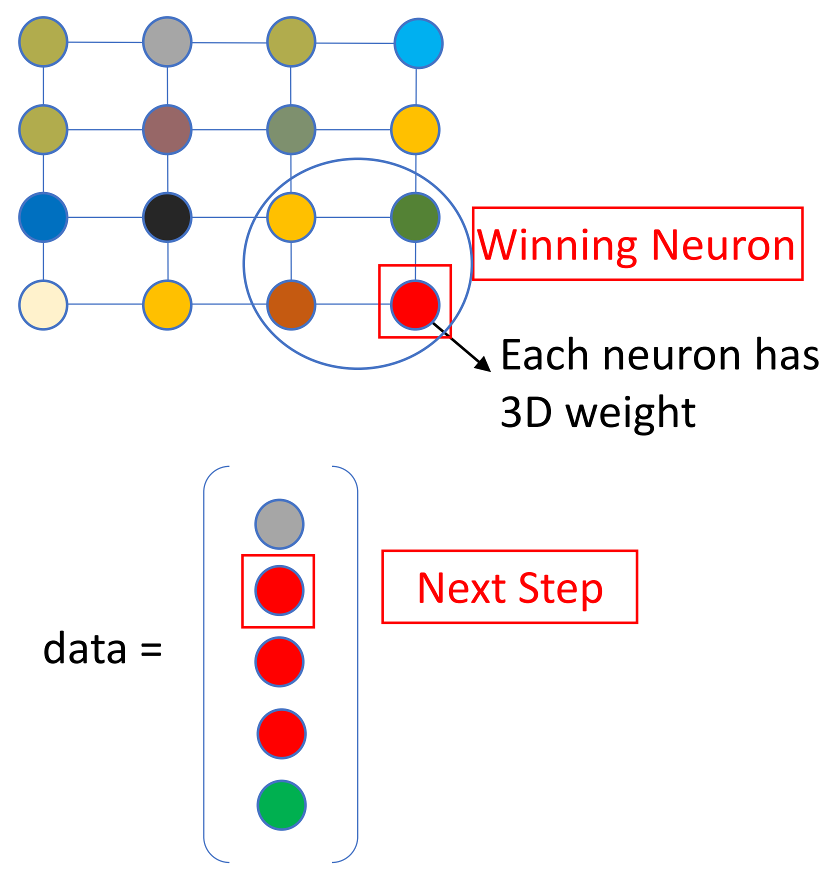
θ_t: Neighborhood function
α_t: Learning Rate

Each neuron has 3D weight

data = 

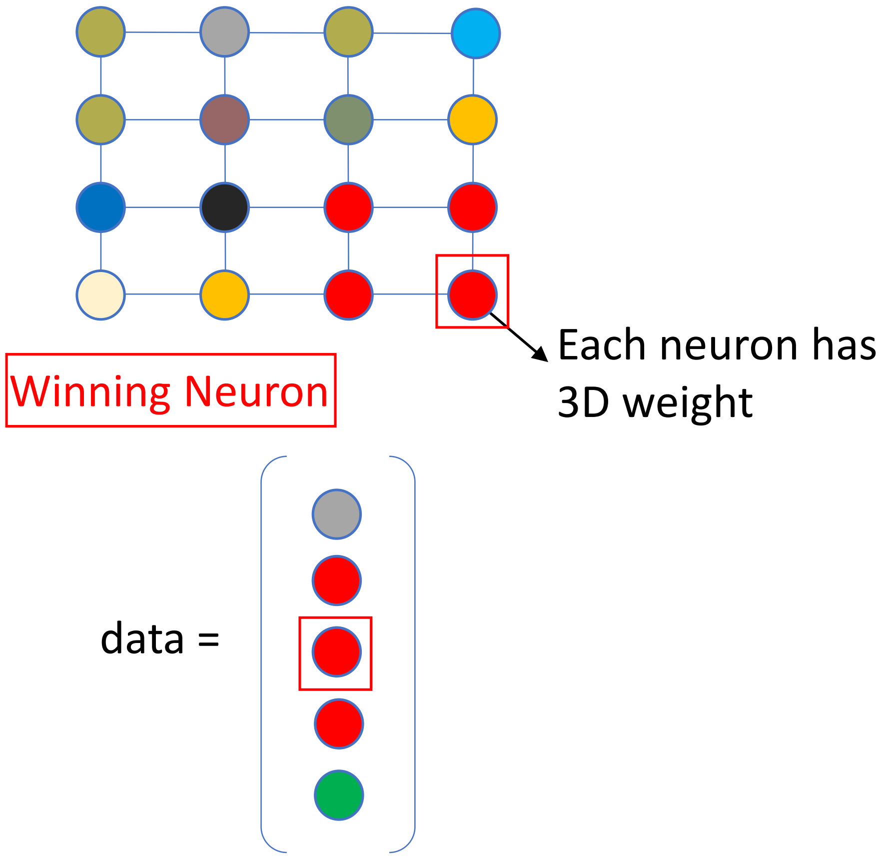
```
1. def som(data):
2.     create a 2D lattice
3.     for d_i in data:
4.         w = find winning neuron in the lattice
5.         update the weights of v^th neuron:
6.             w_v = w_v + θ_t α_t (d_i − w_v)
7.     goto 3. if not converged
```

SOM Algorithm: Loop



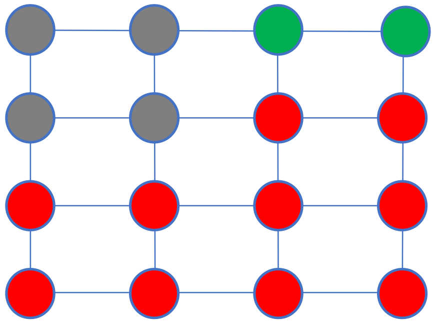
1. **def** som(data):
2.   create a 2D lattice
3.   **for d_i in data:**
4.   **w = find winning neuron in the lattice**
5.   **update the weights of v^{th} neuron:**
6. $w_v = w_v + \theta_t \alpha_t (d_i - w_v)$
7.   **goto** 3. if **not converged**

SOM Algorithm: Loop

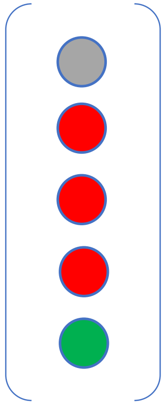


1. **def** som(data):
2.   create a 2D lattice
3.   **for d_i in data:**
4.   **w = find winning neuron in the lattice**
5.   **update the weights of v^{th} neuron:**
6. $w_v = w_v + \theta_t \alpha_t (d_i - w_v)$
7.   **goto** 3. if **not converged**

SOM Algorithm: Final



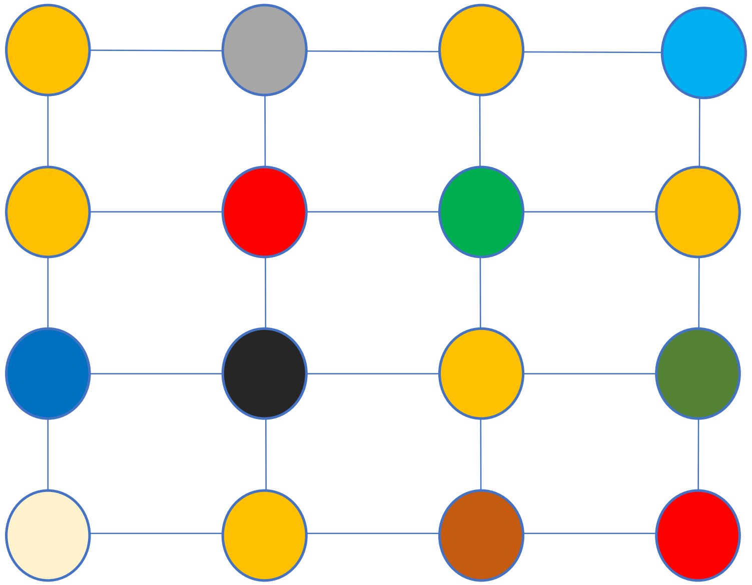
data =



```
1. def som(data):
2.     create a 2D lattice
3.     for d_i in data:
4.         w = find winning neuron in the lattice
5.         update the weights of v^th neuron:
6.             w_v = w_v + θ_t α_t (d_i − w_v)
7.     goto 3. if not converged
```

$$w_v = w_v + \theta_t \alpha_t (d_i - w_v)$$

- How many maximum # of quantized states are possible?



Quantitative example

Training Process

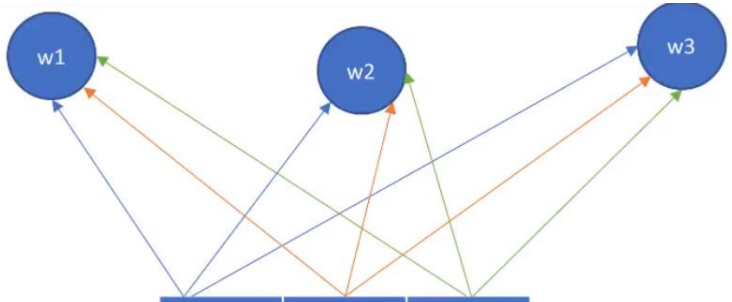
1. Initialize neural network weights

2. Randomly select an input

3. Select the winning neuron using Euclidean distance

4. Update neuron weights

5. Go back to 2 until done training

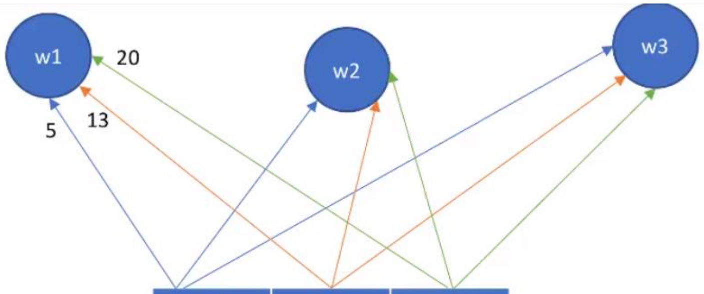


x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

Random initializations

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training

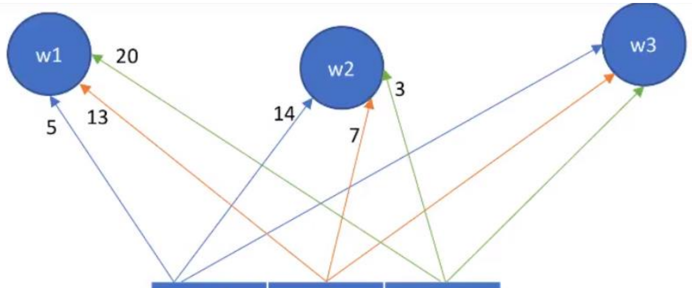


x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

Random initializations

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training

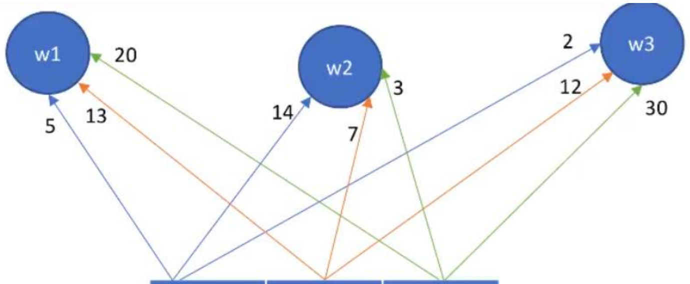


x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

Random initializations

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training

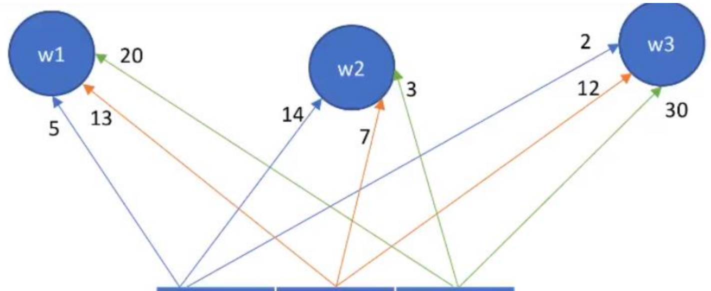


x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

Select an input data randomly

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training

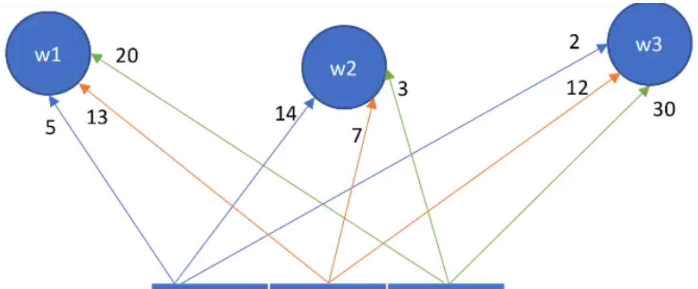


x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

Select the wining neuron

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



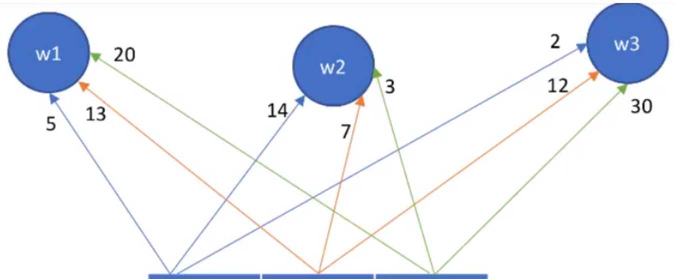
x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

$$d_1 = \sqrt{\sum_{i}^{3} (x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

Select the wining neuron

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

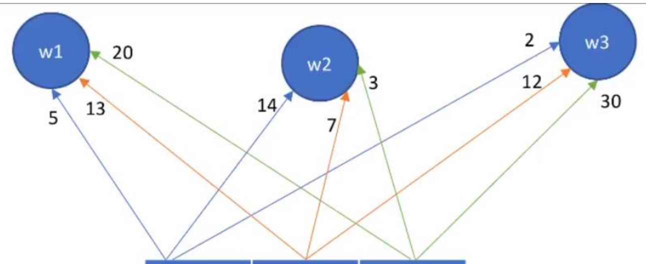
$$d_1 = \sqrt{\sum_{i}^{3}\left(x_i - w_{1,i}\right)^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

$$d_2 = \sqrt{\sum_{i}^{3}\left(x_i - w_{2,i}\right)^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

Select the wining neuron

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

$$d_1 = \sqrt{\sum_{i}^{3}(x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

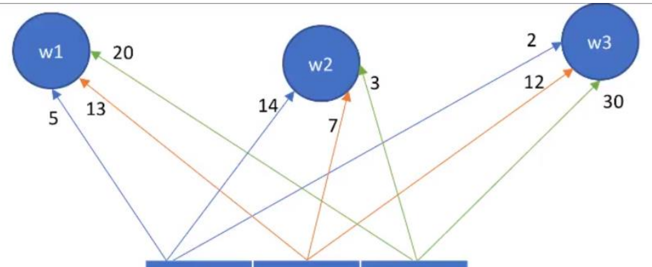
$$d_2 = \sqrt{\sum_{i}^{3}(x_i - w_{2,i})^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

$$d_3 = \sqrt{\sum_{i}^{3}(x_i - w_{3,i})^2} = \sqrt{(0.5-2)^2 + (0.3-12)^2 + (0.7-30)^2} = 31.6$$

Select the wining neuron

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	...

$$d_1 = \sqrt{\sum_i^3 (x_i - w_{1,i})^2} = \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

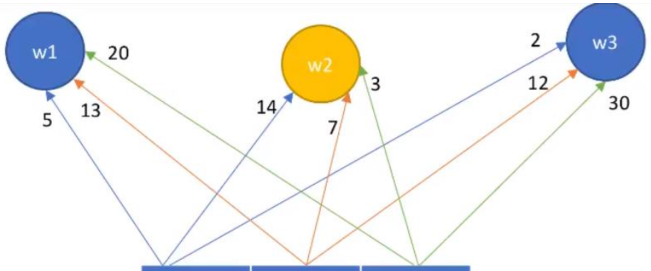
$$d_2 = \sqrt{\sum_i^3 (x_i - w_{2,i})^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

$$d_3 = \sqrt{\sum_i^3 (x_i - w_{3,i})^2} = \sqrt{(0.5-2)^2 + (0.3-12)^2 + (0.7-30)^2} = 31.6$$

Select the wining neuron

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



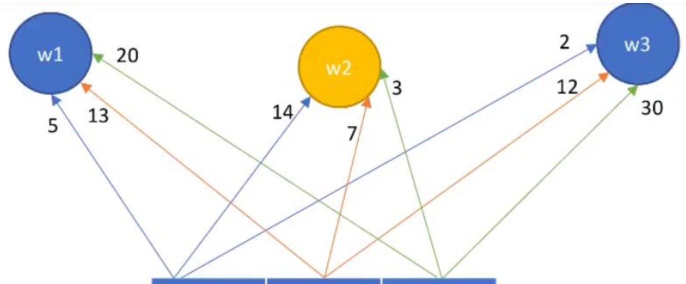
x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	

$$d_2 = \sqrt{\sum_i^3 (x_i - w_{2,i})^2} = \sqrt{(0.5 - 14)^2 + (0.3 - 7)^2 + (0.7 - 3)^2} = 15.2$$

Update neurons weight

Training Process

1. Initialize neural network weights
2. Randomly select an input
3. Select the winning neuron using Euclidean distance
4. Update neuron weights
5. Go back to 2 until done training



x1	x2	x3
0.10	0.20	0.13
0.50	0.30	0.70
...	...	

$$d_2 = \sqrt{\sum_i^3 \left(x_i - w_{2,i}\right)^2} = \sqrt{(0.5-14)^2 + (0.3-7)^2 + (0.7-3)^2} = 15.2$$

$$\boxed{\Delta w_{j,i} = \eta(t) * T_{j,I(x)}(t) * (x_i - w_{j,i})}$$

Update neurons weight

$$\Delta w_{j,i} = \eta(t) * T_{j,I(x)}(t) * (x_i - w_{j,i})$$

Learning Rate: $\eta(t) = \eta_0 \exp\left(-\frac{\mathrm{t}}{\tau_\eta}\right)$

Topological Neighborhood: $T_{j,I(x)}(t) = \exp\left(-\frac{S_{j,I(x)}^2}{2\sigma(t)^2}\right)$

Lateral Distance Between Neurons: $S_{j,i} = \|w_j - w_i\|$

Neighborhood size: $\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0}\right)$

Definitions:

t == epoch

i == a neuron

j == another neuron

I(x) == the winning neuron

Hyperparameters:

η_0

τ_η

σ

τ_0

Lateral distance between neurons

$$\Delta w_{j,i} = \eta(t) * T_{j,I(x)}(t) * (x_i - w_{j,i})$$

Learning Rate: $\eta(t) = \eta_0 \exp\left(-\frac{\text{t}}{\tau_\eta}\right)$

Topological Neighborhood: $T_{j,I(x)}(t) = \exp\left(-\frac{S_{j,I(x)}^2}{2\sigma(t)^2}\right)$

Lateral Distance Between Neurons: $S_{j,i} = \|w_j - w_i\|$

Neighborhood size: $\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0}\right)$

Definitions:

t == epoch

i == a neuron

j == another neuron

I(x) == the winning neuron

Hyperparameters:

η_0

τ_η

σ

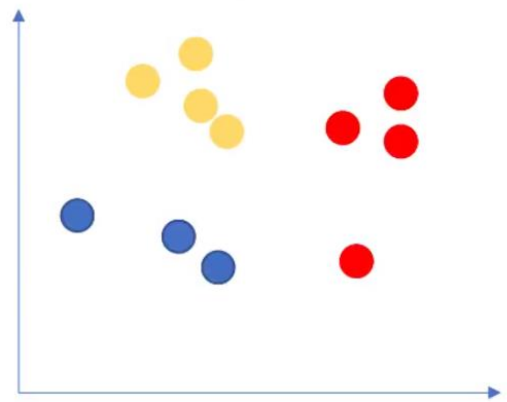
τ_0

How Does Dimensionality Reduction Occur?

N dimensions

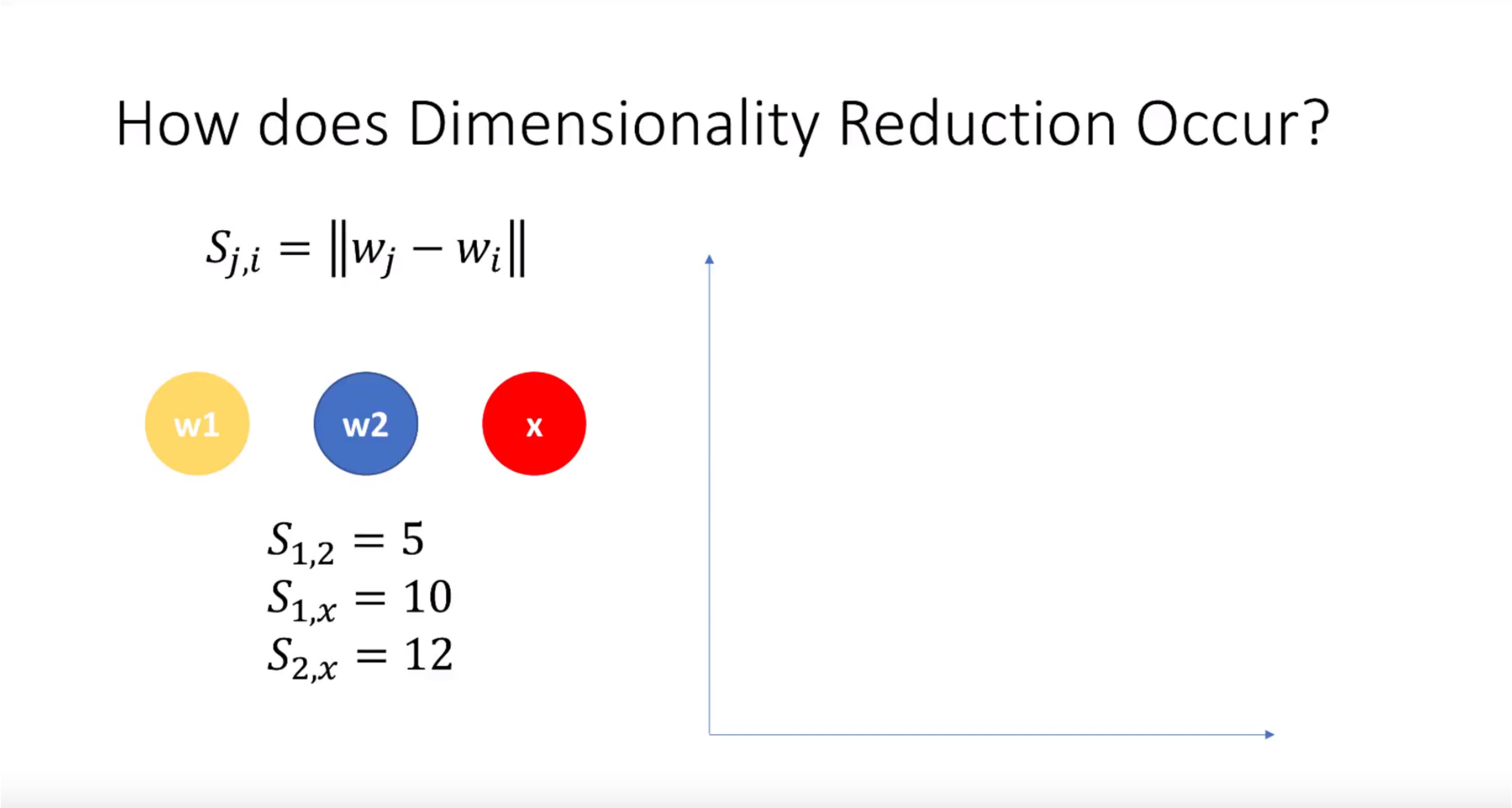
X1	X2	...	XN
0.4	0.5	...	0.8
⋮	⋮	⋮	⋮

2 dimensions

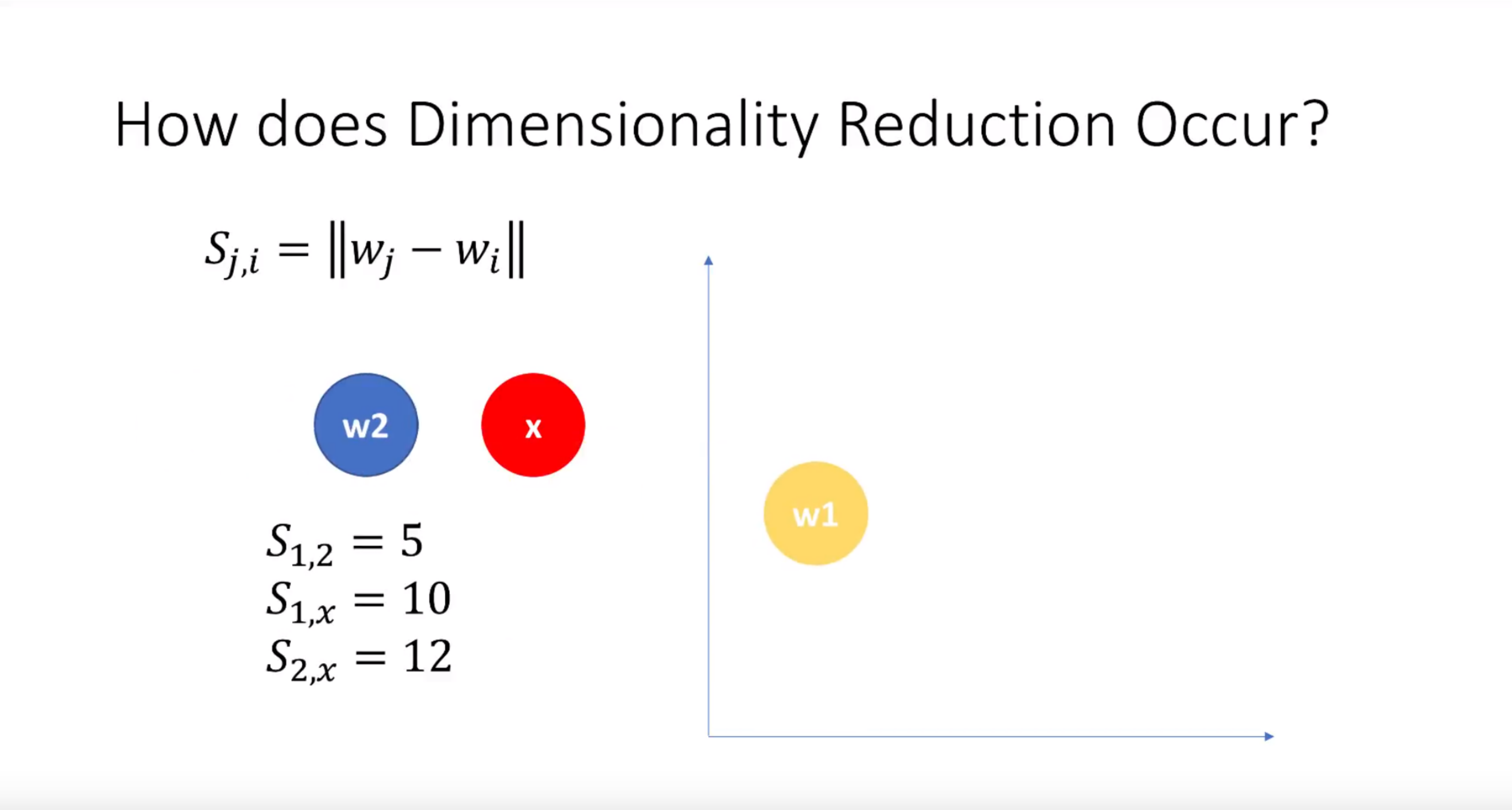


$$S_{j,i} = \|w_j - w_i\|$$

How Does Dimensionality Reduction Occur?



How Does Dimensionality Reduction Occur?



How Does Dimensionality Reduction Occur?

How does Dimensionality Reduction Occur?

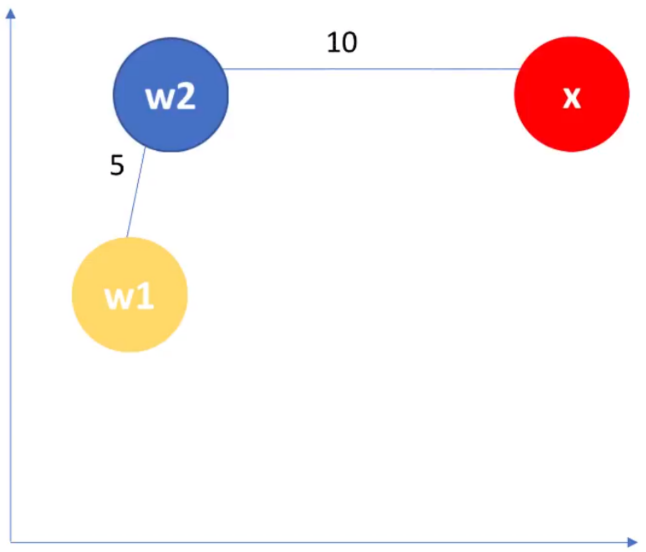
$$S_{j,i} = \|w_j - w_i\|$$

$$S_{1,2} = 5$$
$$S_{1,x} = 10$$
$$S_{2,x} = 12$$

How Does Dimensionality Reduction Occur?

How does Dimensionality Reduction Occur?

$$S_{j,i} = \|w_j - w_i\|$$

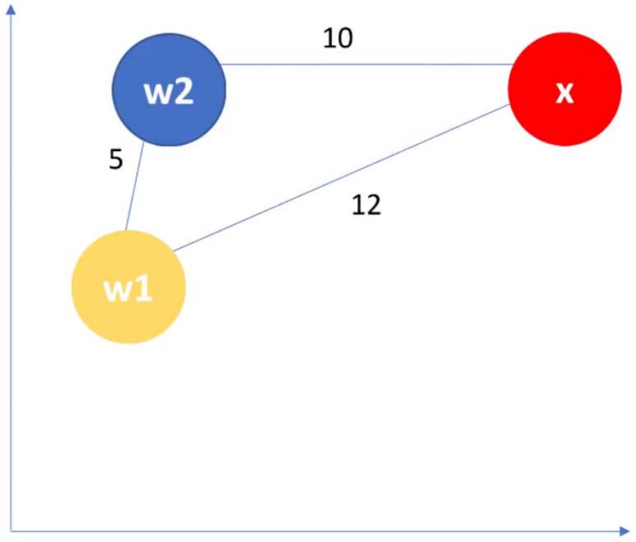


$$S_{1,2} = 5$$
$$S_{1,x} = 10$$
$$S_{2,x} = 12$$

How Does Dimensionality Reduction Occur?

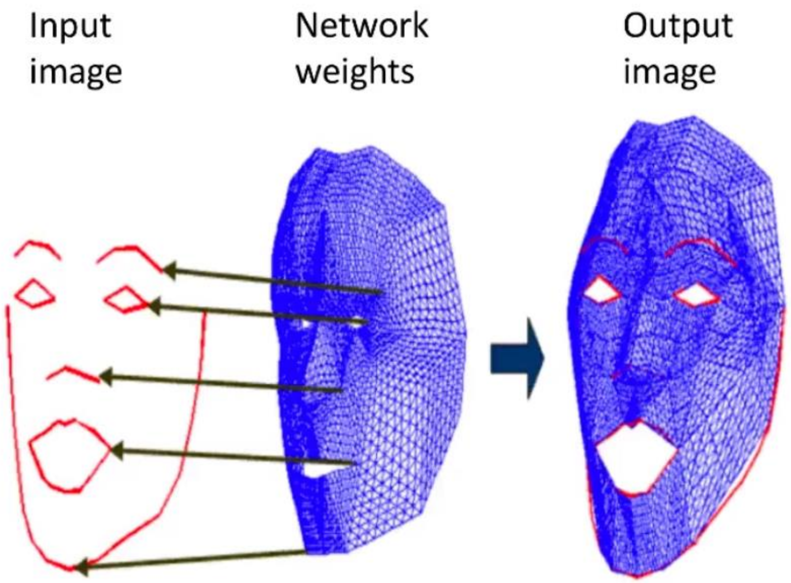
How does Dimensionality Reduction Occur?

$$S_{j,i} = \|w_j - w_i\|$$



$$S_{1,2} = 5$$
$$S_{1,x} = 10$$
$$S_{2,x} = 12$$

- Self-organizing maps were trained on 3D images to construct a 3D head model
- Can be applied to emotion recognition
- The input image gets mapped to clusters in the output space



Sajó, L., Hoffmann, M., Fazekas, A.: A 3D Head Model From Stereo Images by a Self-organizing Neural Network, *Journal for Geometry and Graphics*, **13**, 209-220, 2009

Ref: https://www.youtube.com/watch?v=g8O6e9C_CfY

- Notebook!