

Simulating Neural Networks

Lawrence Ward

P465A



1. Neural network (or **Parallel Distributed Processing, PDP**) models are used for a wide variety of roles, including recognizing patterns, implementing logic, making decisions, etc.

1.1 They rescued artificial intelligence research from the dead end of symbolic AI and are now one part of the foundation of the new AI (the other is agent-based processing).

1.2 **They learn**, making them one of the most powerful tools for psychological modeling.

1.3 **They have limitations**, however, one of which is that representations of psychologically meaningful concepts are difficult to see in the model.

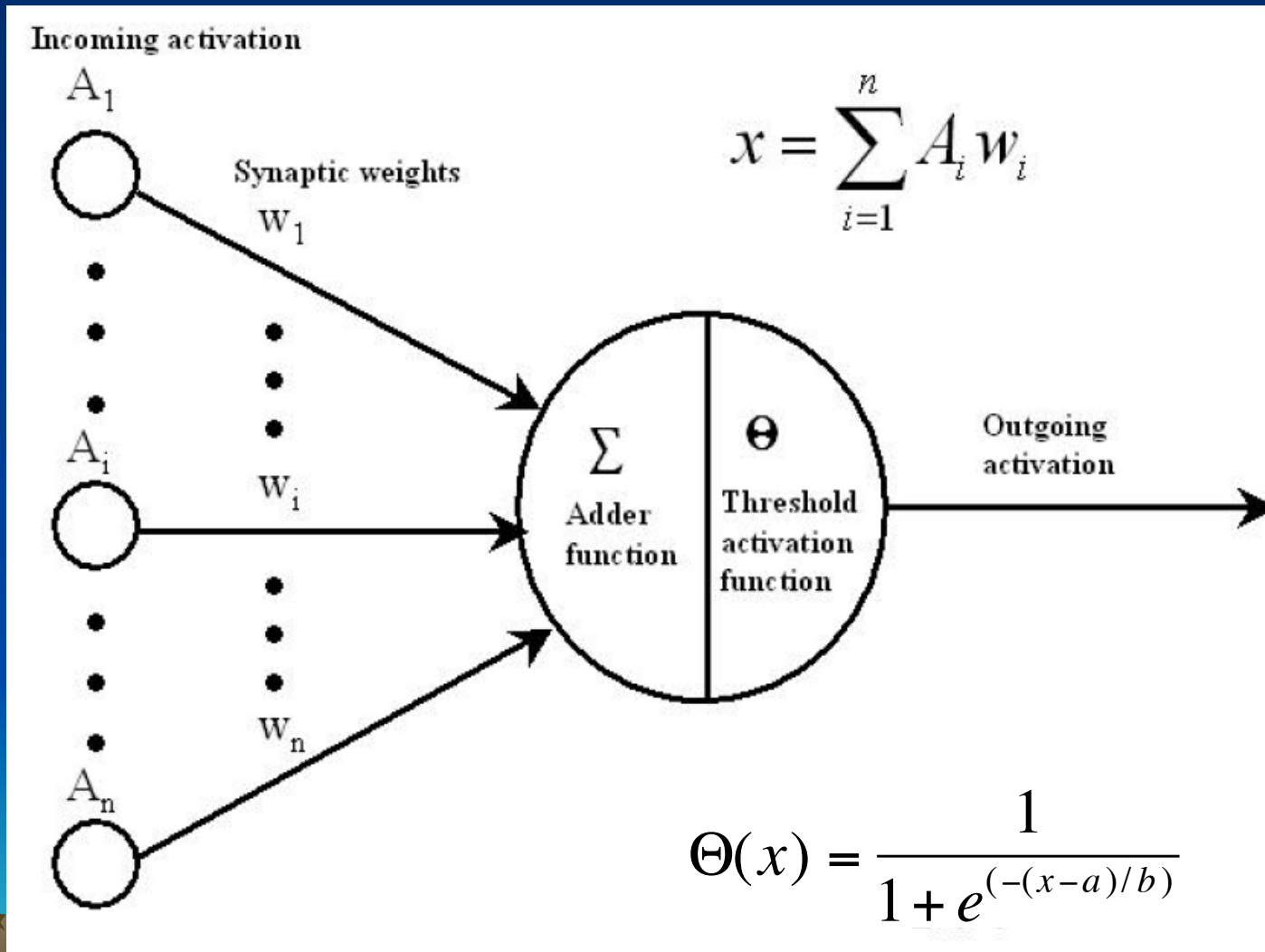
1.4 **They are inherently dynamic** but dynamics is not emphasized (e.g. the learning process) only the outcome (responses after learning).

1.5 They are to be contrasted with **spiking neural networks** (or pulse-coupled neural networks) where **dynamics is the focus**.

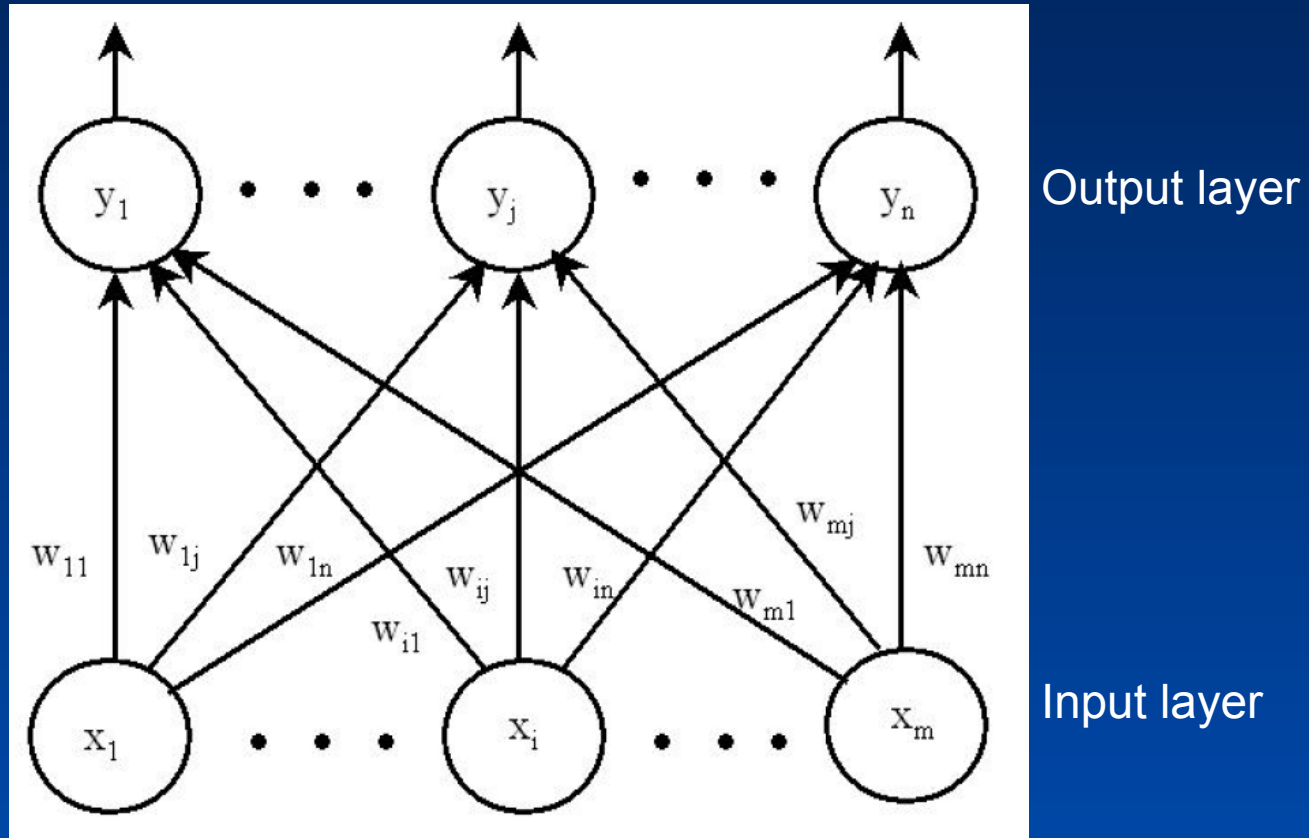
1.6 They implement only some of the many functional characteristics of neurons.



2. The basic model element: the neuron



3. Synapse layer model:



All of the network's "knowledge" is stored in the weights (w_{ij}) of the connections (representing the synapses in a real neural network). We will store the connection weights in a matrix, call it W . For example for a two-layer, 3-element network

$$W = \begin{matrix} & w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{matrix}$$

4. **Learning law**: scheme to adjust the connection weights so that they reflect the desired input-output relation. For example, a network is trained to identify various input patterns by giving a particular output whenever a particular pattern is input.

4.1 If the weight adjustments are based on some abstract rule it is called *unsupervised learning*, but if the weight adjustments are based on some error signal calculated from a desired target output, then it is called *supervised learning*. Supervised learning is most efficient, most reliable, and most used (but unsupervised learning is how humans learn a lot of the time).

4.2 General error correction law:

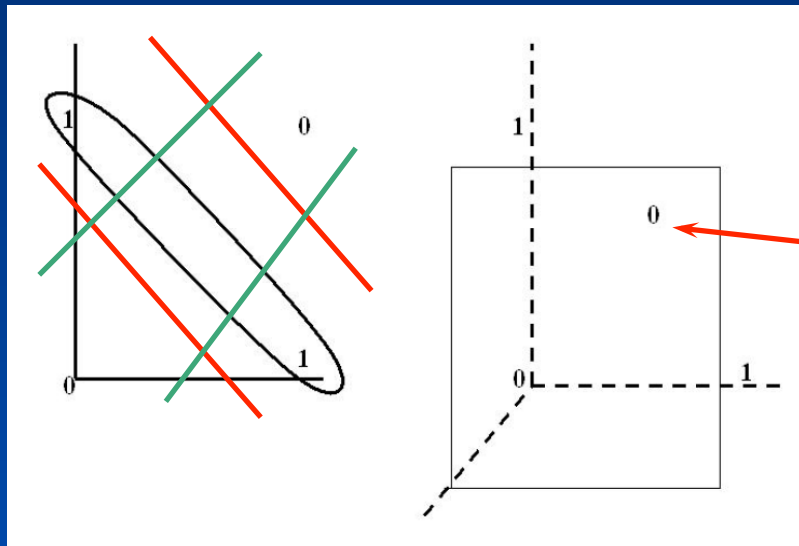
$$\Delta w_{ij} = \alpha a_i (c_j - b_j)$$

where α is the learning rate, a_i is the activation of the i th input-layer neuron, b_j is the activation of the j th output-layer neuron, and c_j is the desired activation of that neuron.

5. Multilayer networks

5.1 Two-layer networks are limited: they cannot learn nonlinear mappings. For example they cannot learn the XOR problem, where the XOR truth table is:

<u>Input1</u>	<u>Input 2</u>	<u>Output</u>
0	0	0
0	1	1
1	0	1
1	1	0

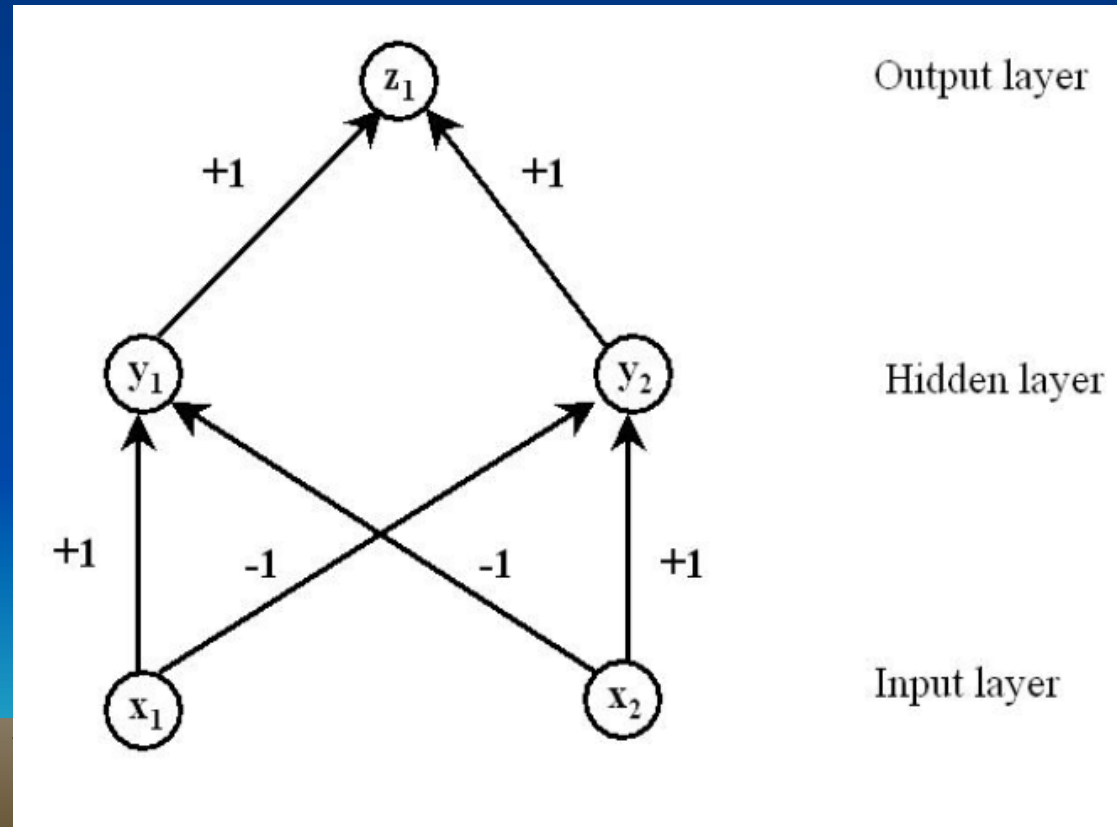


D3 = 1 iff D1=1
and D2=1; plane
is at D3=1

The third dimension is provided by a "hidden layer" that does not directly interact with the outside world. Below is a network that emulates the XOR function. Here the activation function for hidden and output neurons is

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \text{ where } x = \sum_{i=1}^n A_i w_i$$

As an **example**, consider the input $x_1 = 1$ and $x_2 = 0$ respectively. Then for hidden element y_1 the argument for the activation function is $(1 \times 1) + (-1 \times 0) = 1$ and so its activation is $f(1) = 1$. For y_2 it is $(-1 \times 1) + (1 \times 0) = -1$ and so $f(-1) = 0$. Thus the inputs to the output neuron are 1 and 0 making its activation $(1 \times 1) + (1 \times 0) = 1$ and since $f(1) = 1$ its output is 1, as required. Figure out the other possibilities as an exercise.



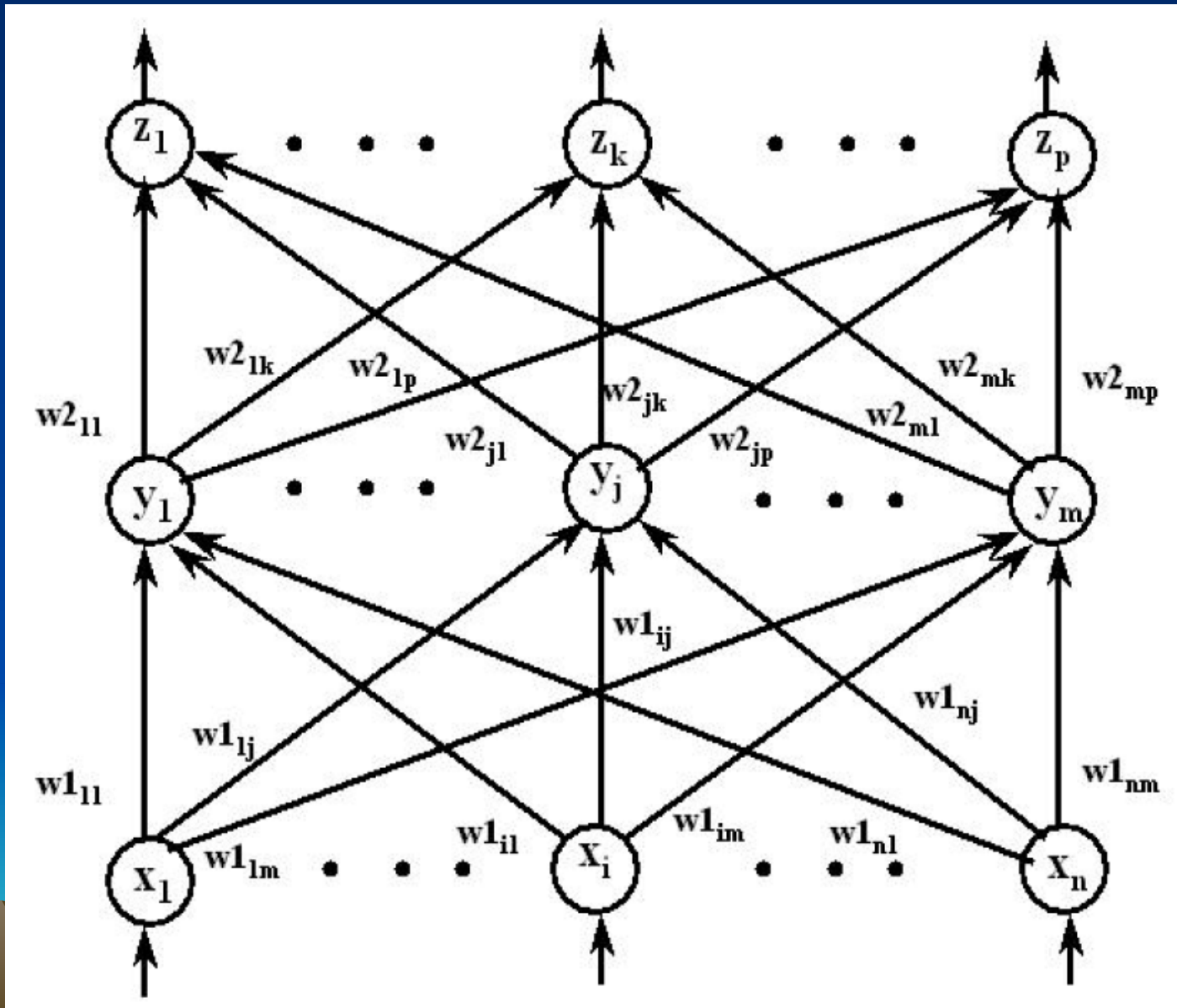
6. Backpropagation saves neural network models

6.1 Minsky & Papert (*Perceptrons, 1969*) criticized multilayer networks because they could see no way to correct the weights of the connections between the input layer and the hidden layer. This and the limitations of 2-layer perceptrons contributed to a long hiatus in neural net research.

6.2 The solution to the problem is to "backpropagate" the error from the output layer to calculate an "error" for each hidden layer neuron: a weighted sum of the output layer errors multiplied by the activation of the hidden-layer neuron. This penalizes the hidden layer neuron in proportion to how much it contributed to the output layer error. These hidden-layer errors are then used to calculate changes to the weights of the input-hidden connections just like is done directly for the hidden-output connections. Output layer error can be backpropagated like this over any number of hidden layers.



7. The basic backpropagation model



7.1 Basic parameters: **weights** (2 layers), **learning rate** $0 < \alpha < 1$, **momentum** (how much a previous weight change affects current weight change) m , **tolerance** (criterion for accepting an output as "good") t .

7.2 Setup: assign random numbers between -1 and +1 to the two sets of weights. Assign values for α , m , and t .

7.3 **Learning forward pass:**

(1) Compute activation for each hidden layer neuron for input pattern selected:

for each hidden neuron summed input is

$$y_j = \sum_{i=1}^n A_i w_{ij}$$

where A_i is the input from the i th input neuron, and the activation is

$$f(y_j) = \frac{1}{1 + e^{-(y_j - a)/b}} = A_j$$

with $a = 0.5$ and $0 < b < 0.5$.

(2) Compute activation for each output layer neuron

$$z_k = \sum_{j=1}^m A_j W_{jk}$$

where $A_j = f(y_j)$ is the input from the j th hidden layer neuron, and the output activation is

$$f(z_k) = \frac{1}{1 + e^{-(z_k - a)/b}} = A_k$$

This vector of output layer neuron activations is the output you check for "goodness."

7.4 Backpropagation of error:

(1) compute the output-layer error:

$$d_k = A_k (1 - A_k) (T_k - A_k)$$

where A_k is the activation of the k th neuron in the output layer.

(2) compute the hidden layer error:

$$e_j = \sum_{k=1}^p A_j (1 - A_j) W_{jk} d_k$$

(3) adjust the weights for the hidden-output synapses

$$W2_{jkt} = W2_{jk(t-1)} + \Delta W2_{jkt}$$

Where

$$\Delta W2_{jkt} = \alpha A_j d_k + m \Delta W2_{jk(t-1)}$$

(4) adjust the weights for the input-hidden synapses

$$W1_{ijt} = W1_{ij(t-1)} + \Delta W1_{ijt}$$

Where

$$\Delta W1_{ijt} = \alpha A_i e_j + m \Delta W1_{ij(t-1)}$$

(5) Repeat until $|T_k - A_k| < \epsilon_k$ for all output neurons and all input patterns.



7.5 **Recall**: repeat the steps for the forward pass. The computed output is the recalled pattern.

7.6 **General approach**: train on clear input patterns, test on same input patterns, test on variants of input patterns.



The assignment

- 2 hidden unit XOR
- 4 hidden unit XOR
- Discuss
 - Effect of number of hidden units on learning rate
 - Differences and similarities of two networks in terms of patterns of synaptic weights between input and hidden, and hidden and output layers in them

The program

Define variables (arrays easiest): alpha, tol, M

Input patterns (x)

	j	
i	0	0
	0	1
	1	0
	1	1

patterns(pindex(i),j)

target outputs

0	1	1	0
---	---	---	---

pattern index

0	1	2	3
---	---	---	---

hbias

0	0	0	0
---	---	---	---

W1 (input->hidden)

	j	
	0	-.50
	.2	1
	.68	-.37
	-.33	.92

W2 (hidden->output)

	j
	.04
	.57
	-.98
	.38

Hidden activation (y , by neuron)

0	0	0	0
---	---	---	---

Output activation (z , by pattern)

0	0	0	0
---	---	---	---

d (by pattern)

0	0	0	0
---	---	---	---

e (by hidden)

0	0	0	0
---	---	---	---

$\text{del}W2$

0	0	0	0
---	---	---	---

$\text{del}W2\text{old}$

0	0	0	0
---	---	---	---

$\text{del}W1$

0	0
0	0
0	0
0	0

$\text{del}W1\text{old}$

0	0
0	0
0	0
0	0



