Artificial Neural Networks

Ottokar Tilk

2014

Lecture scope

- Why should you care;
- Multilayer feedforward ANNs (Multilayer perceptrons);
- Training MLPs with backpropagation.

Why should you care?

From last lecture:

Perceptrons:

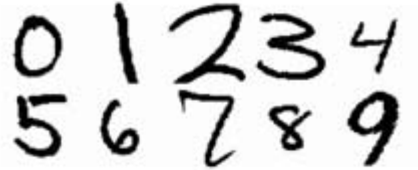
- Can't solve linearly inseparable problems (XOR);
- No probabilistic outputs;
- No multiclass classification.

MLPs can solve these problems.

What can ANNs do?

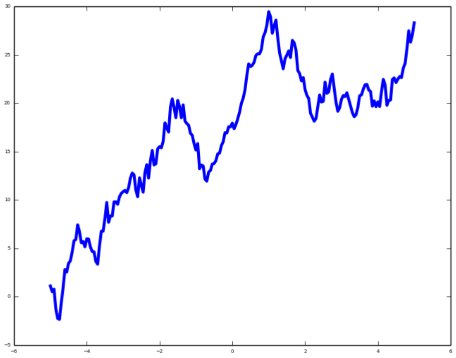
Classification (pattern recognition):

- Object recognition;
- Speech recognition;
- Handwritten text recognition.



Regression (function approximation):

- Stock market prediction.



… and more.

ANNs as the state of the art

- Image classification (MNIST, CIFAR etc...);
- Speech recognition.

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

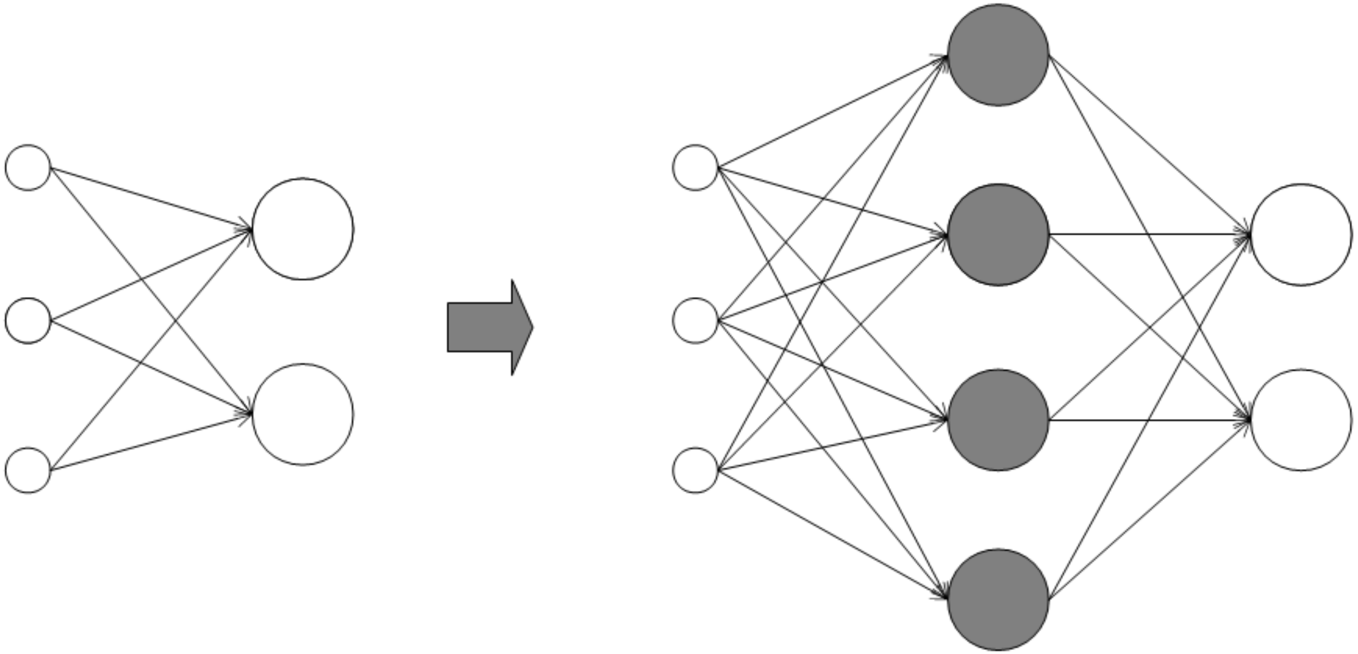
Multilayer perceptrons

Multilayer perceptrons

1.) Multiple layers;
2.) Feedforward;
3.) Nonlinear activations.

Multilayer perceptrons

1.) Multiple layers;



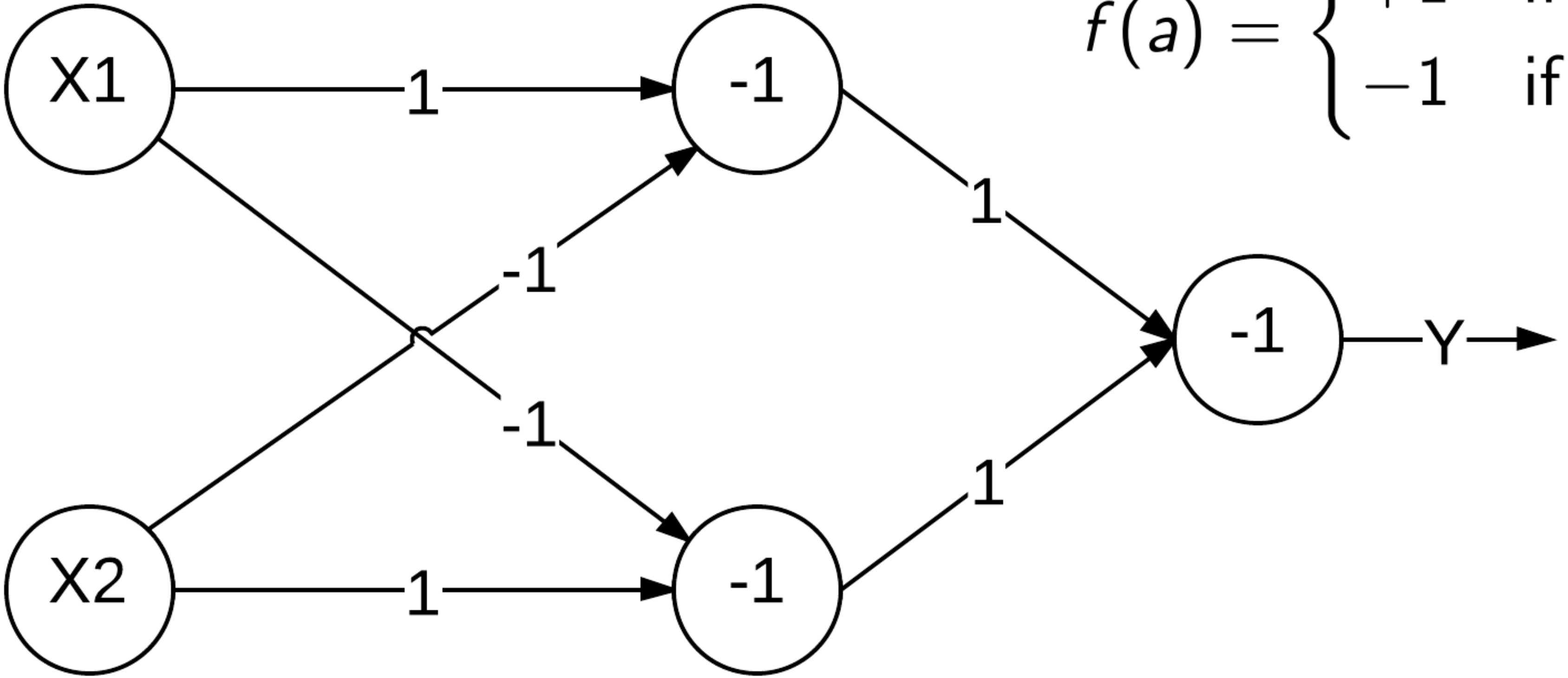
Why more than one layer?

- Necessary to solve the linear separability problem;
- More powerful models (Deep learning).

XOR network

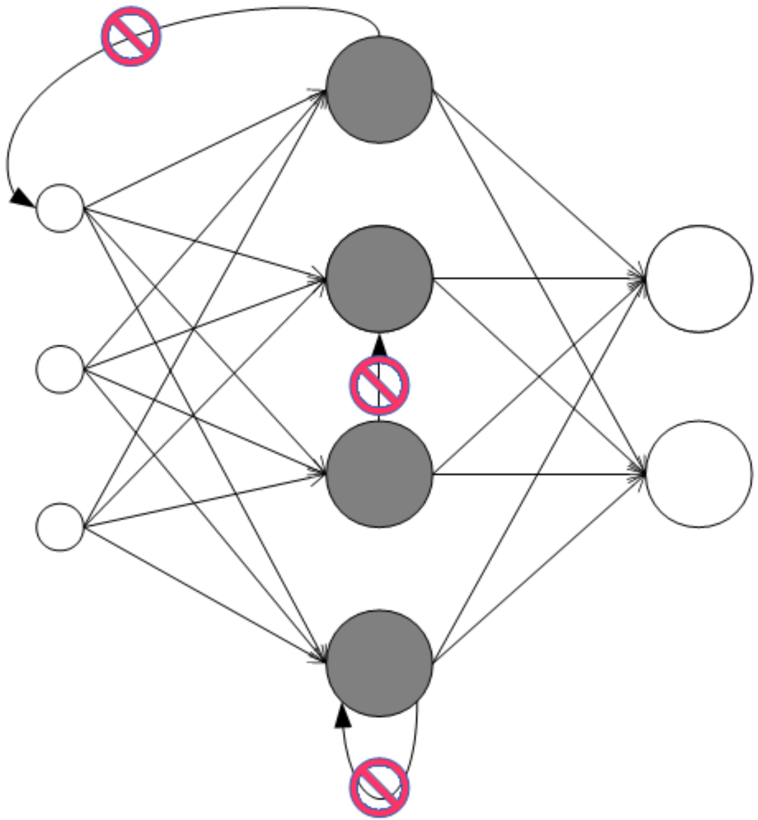
$$a = \sum_{j=1}^{d} w_j x_j + b = \mathbf{w}^T \mathbf{x} + b$$

$$f(a) = \begin{cases} +1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}$$



Multilayer perceptrons

2.) Feedforward;



Why feedforward?

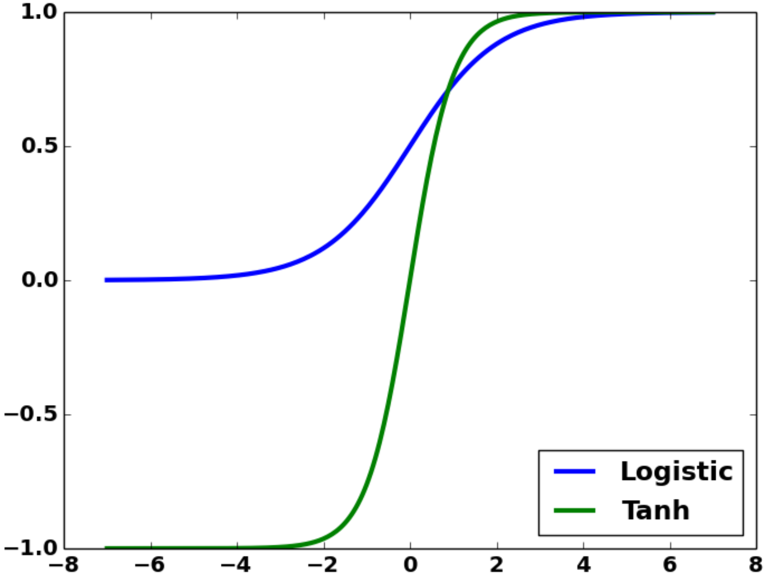
- That’s the definition;
- There are also:
  - Recurrent neural networks;
  - Competitive networks;
  - etc ...

Multilayer perceptrons

3.) Nonlinear activations.

$$f(z) = \frac{1}{1 + e^{-z}}$$

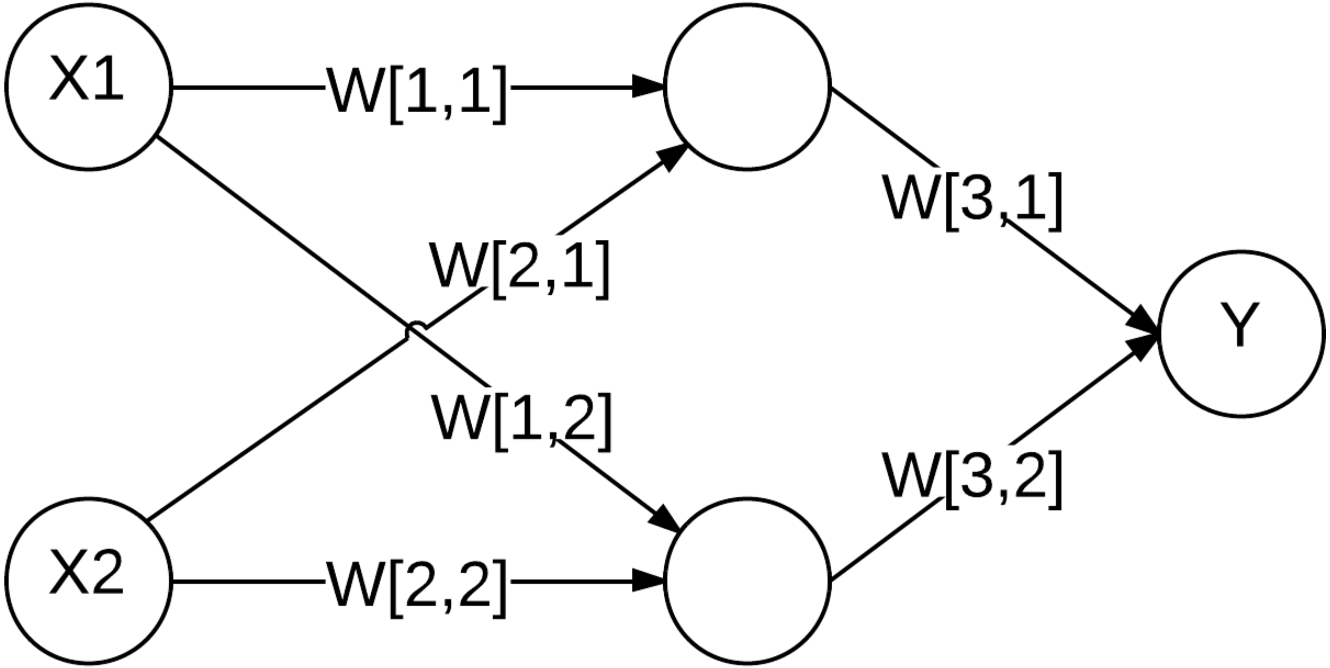
$$f(z) = tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



Why nonlinear activations?

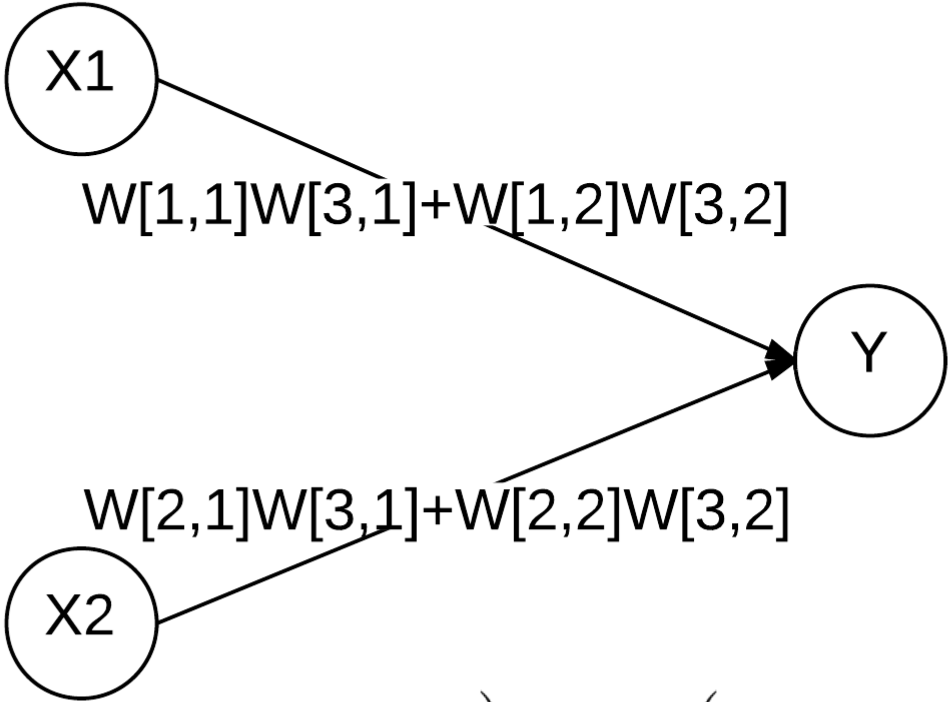
- Otherwise reduces to single layer

Linear multilayer example



$$y = (x_1 w_{1,1} + x_2 w_{2,1}) w_{3,1} + (x_1 w_{1,2} + x_2 w_{2,2}) w_{3,2}$$

Equivalent single layer network



$$y = x_1(w_{1,1}w_{3,1} + w_{1,2}w_{3,2}) + x_2(w_{2,1}w_{3,1} + w_{2,2}w_{3,2})$$

Softmax activation function

- Elements sum to 1;
- All elements $0 < y(i) < 1$;
- Output is a discrete probability distribution;
- Used in multiclass classification;
- A generalization of the logistic function.

$$y(i) = \frac{e^{z(i)}}{\sum_{j=1}^{k} e^{z(j)}}$$

Training MLPs with backpropagation

Backpropagation

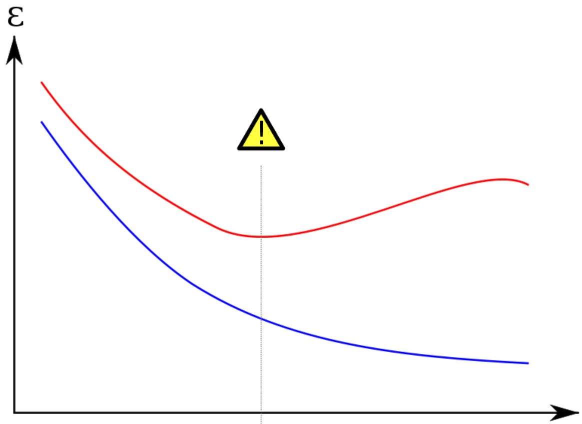
- Supervised;
- Popular;
- Can find local minimum instead of global;
- May converge slowly or not at all.

Before training a MLP

- Split data into training, **validation** and test set;
- Set hyperparameters:
  - Number of layers and number of units per layer;
  - Learning rate;
- Choose error and activation functions.

Validation set

- Not used for training;
- Better estimator of real performance;
- Used for:
  - finding suitable hyperparameters;
  - early stopping to prevent overfitting.



http://en.wikipedia.org/wiki/Overfitting

Error functions

Classification:

- Cross entropy (with softmax)

$$E = -\sum_{i=1}^{k} t_i \ln y_i$$

Regression:

- Summed squared error

$$E = \frac{1}{2}\sum_{i=1}^{k}(t_i - y_i)^2$$

MLP training algorithm

Data: training- and validationData of input-target pairs;
learningRate, layerSizes
Result: trained network

net $\leftarrow$ `InitializeNetwork(`*layerSizes*`)`;
validationError $\leftarrow \infty$;

for *epoch* $\leftarrow 1$ **to** *maxEpochs* **do**
  oldNet $\leftarrow$ net;
  oldValidationError $\leftarrow$ validationError;

  net $\leftarrow$ `Train(`*net, trainingData, learningRate*`)`;
  validationError $\leftarrow$ `Validate(`*net, validationData*`)`;

  if $validationError > oldValidationError$ **then**
    return oldNet;
  end
end
return net;

Training

```
Function Train(net, trainingData, learningRate)
    foreach input-target vector pair (x, t) in trainingData do
        /* y is an array of layer state vectors                    */
        y ← ForwardPropagate(net, x);
        gradients ← BackPropagate(net, x, t, y);
        net ← Update(net, gradients, learningRate);
    end
    return net;
```

Validation

```
Function Validate(net, validationData)
    totalError ← 0;
    foreach input-target vector pair (x, t) in validationData do
        y ← ForwardPropagate(net, x);
        totalError ← totalError + Error(y, t);
    end
    averageError ← totalError / number of validation samples;
    return averageError;
```

Forward propagation

Function `ForwardPropagate(`net, x`)`

$y \leftarrow []$;

$k \leftarrow$ number of layers;

$y_0 \leftarrow x$;

for $i \leftarrow 1$ **to** k **do**

`/* W and f are layer weights and activation function */`

$z_i \leftarrow y_{i-1} W_i$;

$y_i \leftarrow f_i(z_i)$;

$y \leftarrow [y, y_i]$;

end

return y;

Backpropagation

Function `BackPropagate`(net, x, t, y)

gradients $\leftarrow$ [];

$k \leftarrow$ number of layers;

$y_0 \leftarrow x$;

$\frac{\partial E}{\partial y_k} \leftarrow$ `ErrorDerivative`(t, y_k);

for $i \leftarrow k$ **to** *1* **do**

$\frac{\partial E}{\partial z_i} \leftarrow \frac{\partial E}{\partial y_i} * \frac{dy_i}{dz_i}$;

$\frac{\partial E}{\partial W_i} \leftarrow y_{i-1}^T \frac{\partial E}{\partial z_i}$;

gradients $\leftarrow [\frac{\partial E}{\partial W_i}$, gradients];

if $i > 1$ **then**

$\frac{\partial E}{\partial y_{i-1}} = \frac{\partial E}{\partial z_i} W_i^T)$;

end

end

return gradients;

Updating the model

Function `Update`$(net,\ gradients,\ learningRate)$

$k \leftarrow$ number of layers;

for $i \leftarrow 1$ **to** k **do**

$W_i \leftarrow W_i - learningRate \frac{\partial E}{\partial W_i}$;

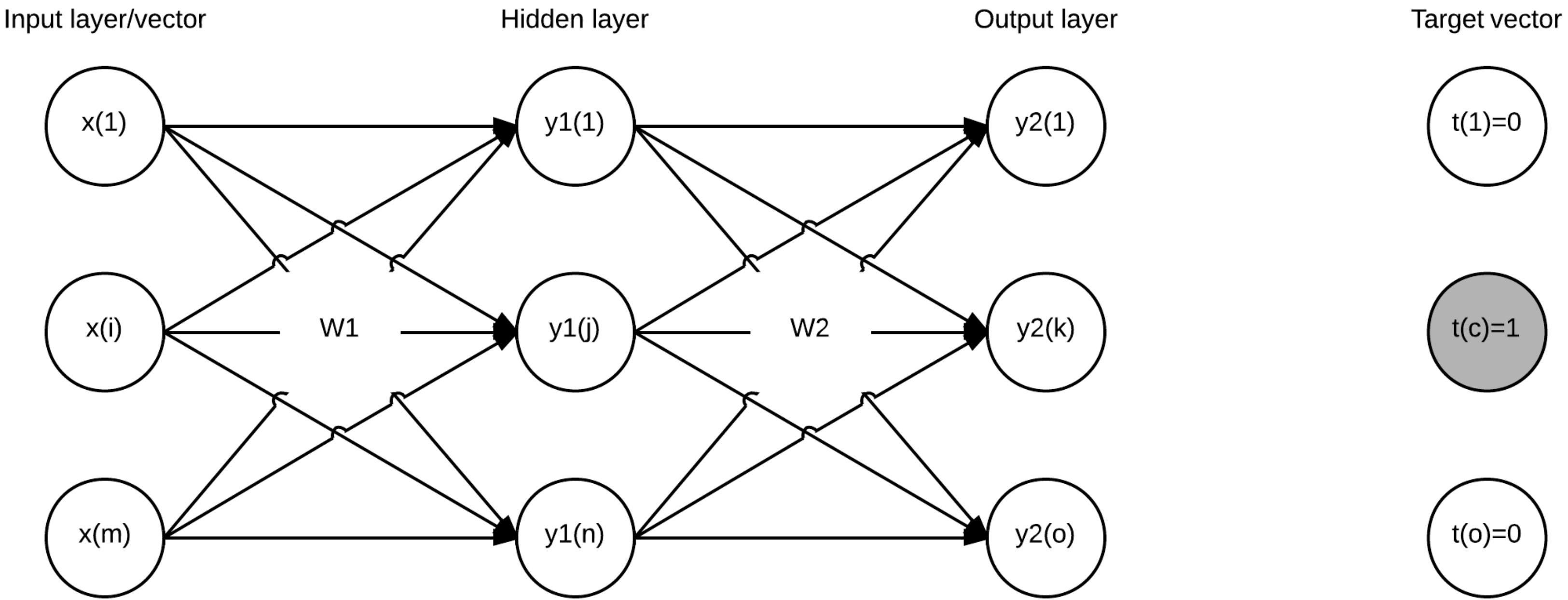
end

return net;

Example

$$z_1(j) = \sum_{i=1}^{m} x(i) W_1(i,j) \qquad z_2(k) = \sum_{j=1}^{n} y_1(j) W_2(j,k)$$

$$y_1(j) = \frac{1}{1 + e^{-z_1(j)}} \qquad y_2(k) = \frac{e^{z_2(k)}}{\sum_{l=1}^{o} e^{z_2(l)}} \qquad E = -\ln y_2(c)$$



Thank you!