Neural Networks and HPC synergy

Raphael Cóbe
Rogério Iope
Silvio Stanzanni
Jefferson Fialho
{rmcobe,rogerio,silvio,jfialho}@ncc.unesp.br

Machine Learning

- an algorithm that is able to learn from data;
- what do we mean by learning?

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

– Tom Mitchel

Machine Learning

- allows us to tackle tasks that are too difficult to solve with fixed programs
- process examples;
- tasks:
  - classification, clustering, regression, translation, …

Machine Learning

- the experience:
  - unsupervised
    - inference from data structure
  - supervised
    - learn by examples;

Machine Learning

- The more data, the best!
  - Overfitting and Underfitting;
- Common solutions:
  - linear regression, k-means, logistic regression, support vector machines, etc…

Neural Networks

Neural Networks

Neurons

- Receive input from other units and decides whether or not to fire.
- ~ 86 billion neurons in the human nervous system
  - connected with approximately 10^14 - 10^15 synapses

Neural Networks

Neurons

- input signals from its dendrites;
- output signals along its (single) axon;
- interact multiplicatively (e.g. $w_0 x_0$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g. w_0);
- learn synapses strengths;

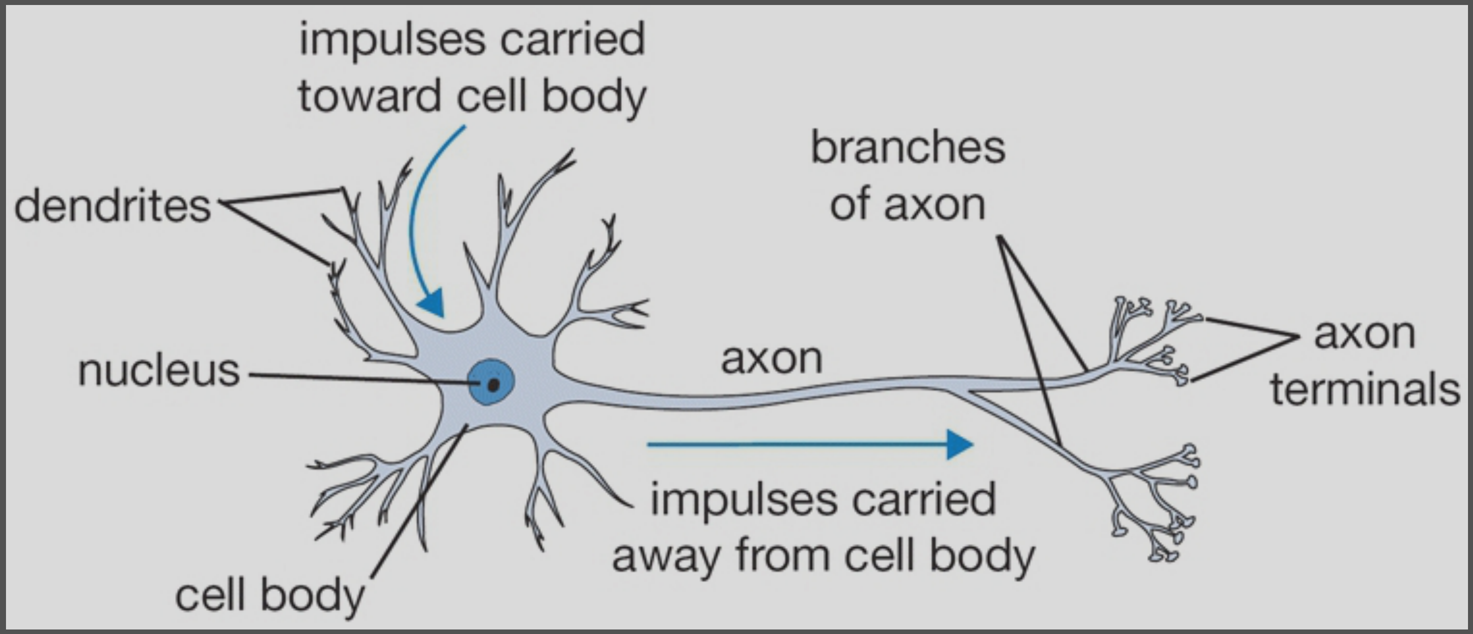
Neural Networks

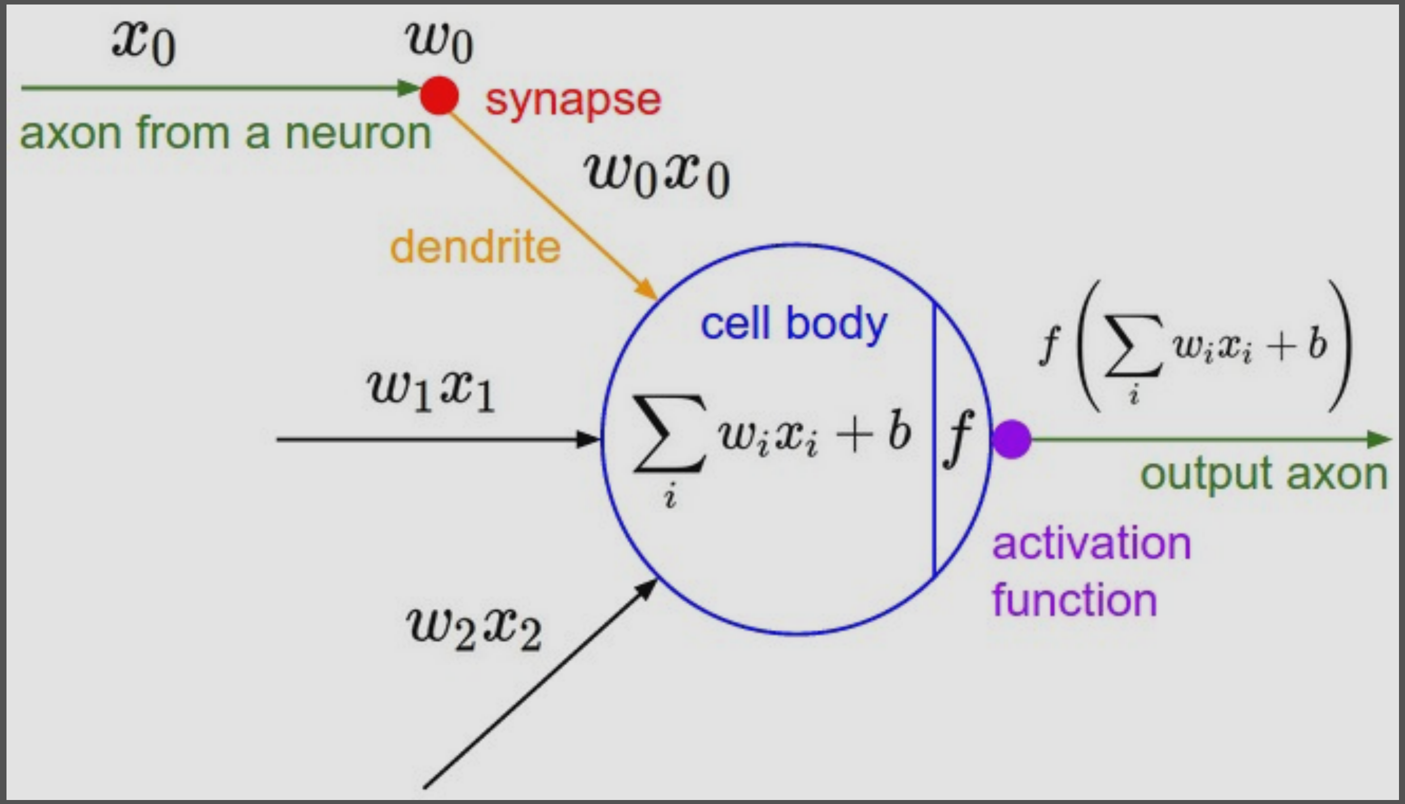
Neurons

- control the influence from one neuron on another:
  - excitatory when weight is positive; or
  - inhibitory when weight is negative;
- nucleus is responsible for summing the incoming signals;
  - if the sum is above some threshold, then fire!

Neural Networks

Neurons





Images from Stanford CS231

Neural Networks

- Function approximation machines;
- $y = f^*(x)$
  - maps x input to a y category
- $y = f(x; w)$
  - learn the value of the w parameters

Neural Networks

- Input, Output, and Hidden layers;
- Hidden as in "not defined by the output";
- Approximate $y = f(x; w)$ to $y = f^*(x)$ (training)

Neural Networks

Hidden Layers

- Seen as vector-valued, i.e. a vector is received as input and a new vector is produced as output;
  - (vector-to-vector function);
- Units that work in **parallel**.
  - (vector-to-scalar function);

Neural Networks

- Activation Function:
  - Describes whether or not the neuron fires, i.e., if it forwards its value for the next neuron layer;
- multiply the input by its weights, add the bias and apply the non-linearity;
- Sigmoid, Hyperbolic Tangent, Rectified Linear Unit;

Neural Networks

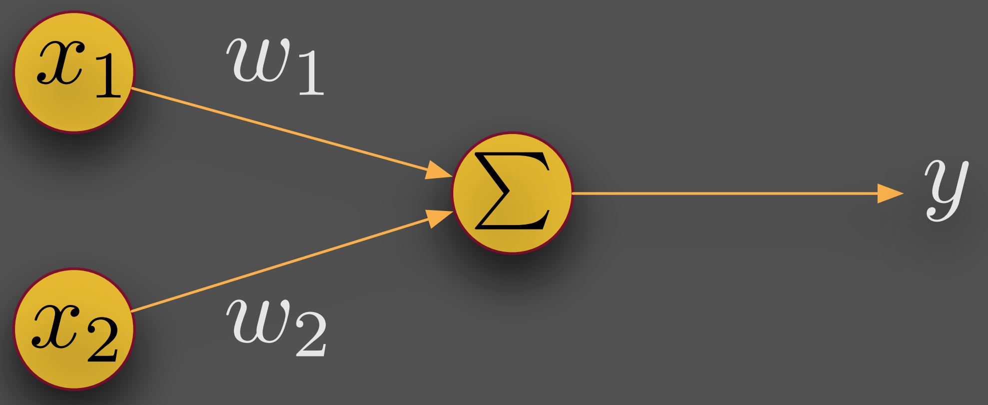
```
class Neuron(object):
  def forward(inputs):
    """ assume inputs and weights are 1-D
    numpy arrays and bias is a number """

    cell_body_sum = np.sum(inputs * self.weights) + self.bias

    # ReLU activation function
    firing_rate = np.maximum(cell_body_sum, 0, cell_body_sum)
    return firing_rate
```

Perceptron

- In 1958, Frank Rosenblatt proposed an algorithm for training the perceptron.
- Simplest form of Neural Network;
- One unique neuron;
- Adjustable Synaptic weights;
- Simple activation function;

Perceptron

- Classifies inputs into two classes, with neuron output of either -1 or 1;



Perceptron

- Where ϕ is the activation function and J_1 the number of inputs

- Always converge on linearly separable classes;

$$net = \sum_{i=1}^{J_1} w_i x_i - \theta = w^T x - \theta$$

$$y = \phi(net)$$

Perceptron

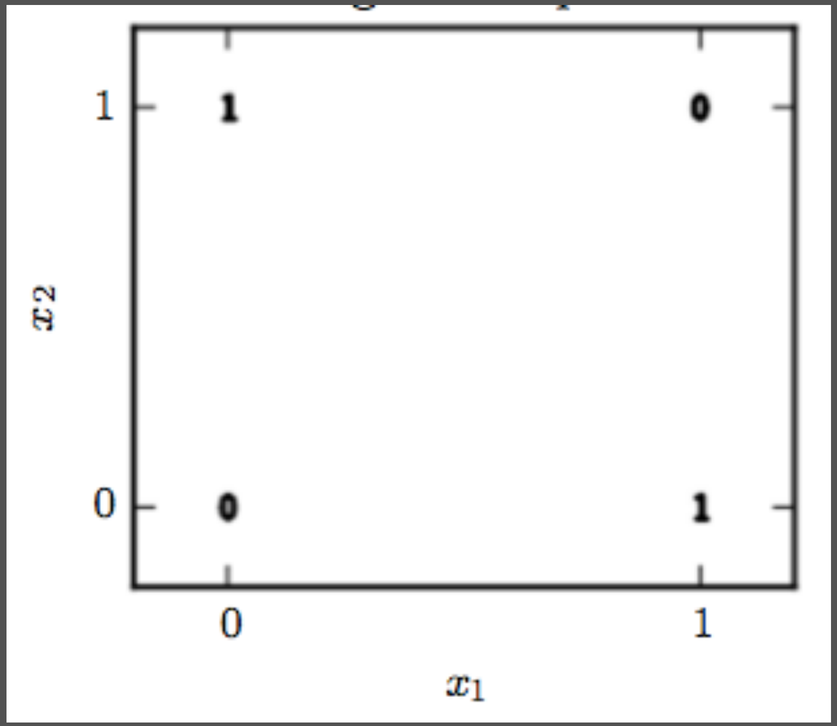
- Training Procedure:
- η is the learning rate;
- finds a linear function that separates the classes;

$$e_p = y_p - \hat{y}_p$$

$$w_i(t+1) = w_i(t) + \eta x_{p,i} e_i$$

Perceptron

- Can't do:
  - separate non linear classes;
  - XOR function:



Feedforward Networks

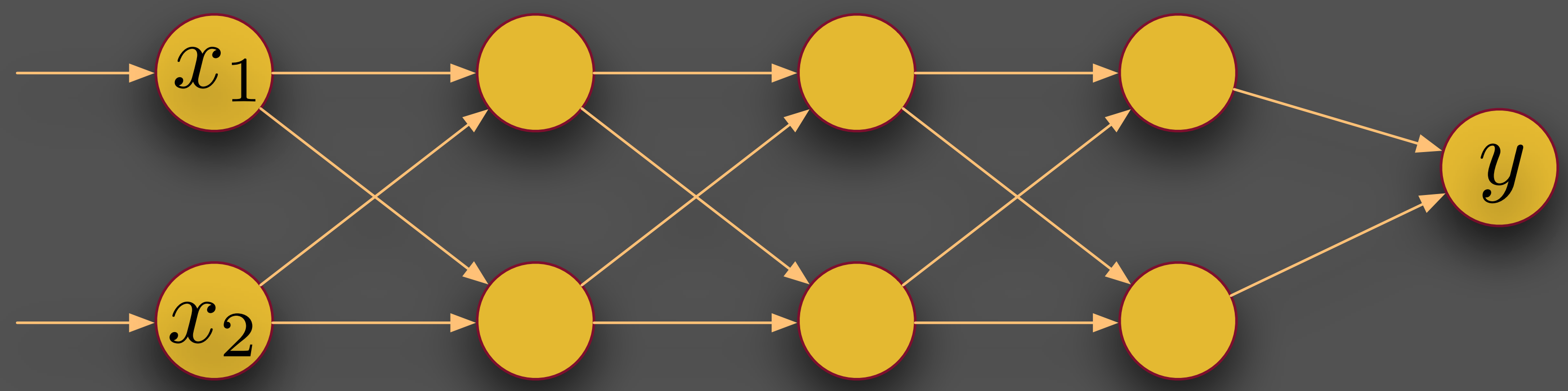
- typically represented by composing many different functions:

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

- the depth of the network - the deep in deep learning! (-;

Feedforward Networks

- Information flows from x, through f computations and finally to y
- No feedback!



Feedforward Networks

Hidden Layers

- Train using a back-propagation algorithm from 1969;
  - fixes the weights in an output-to-input direction;
- each level has plays a specific role in the classification;
- detect the features in the input patterns;

Feedforward Networks

How it works

- The output of a Feedforward Network:

$$\hat{y}_p = o_p^{(M)},\, o_p^1 = x_p$$

- The output of the m layer:

$$net_p^{(m)} = [W^{(m-1)}]^T o_p^{(m-1)} + \theta^{(m)}$$

$$o_p^{(m)} = \phi^{(m)}(net_p^{(m)})$$

Feedforward Network

XOR example

- suppose we want to learn the behavior of the binary XOR operator:
- the input will be a pair of signals with value either 0 or 1.
- The output should be classified also into 0 or 1;

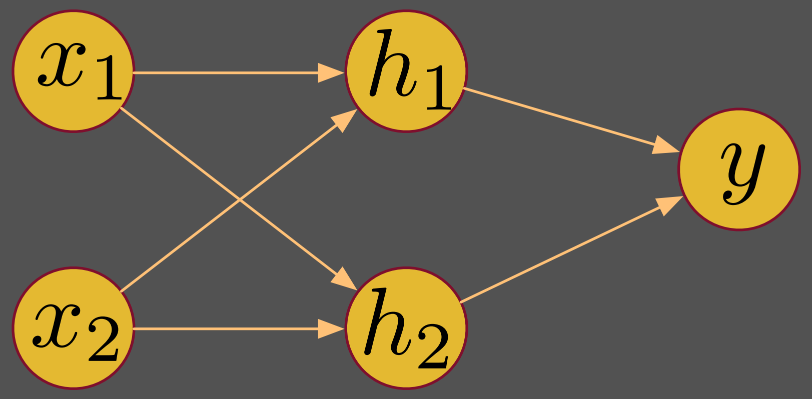
Feedforward Network

XOR example

- The dataset:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

The network:



Feedforward Network

XOR example

- The architecture: 2 layer network (one hidden and one for output)
- Rectified linear Unit as activation function, i.e. $\phi(net_p^{(1)}) = max(net_p^{(1)}, 0)$
- Why not a linear function?

Feedforward Network

XOR example

- The full expanded solution:

$$y = W_2^T(max(0, W_1^T x + \theta_1)) + \theta_2$$

Feedforward Network

XOR example

- Running the example:

$$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \theta_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad \theta_2 = 0$$

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Feedforward Network

XOR example

- Running the example:

$$net^{(1)} = W_1 X + \theta_1 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\phi(net^{(1)}) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

Feedforward Network

XOR example

- Running the example:

$$o^{(2)} = net^{(2)} = W_2 o^{(1)} + \theta_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Feedforward Networks

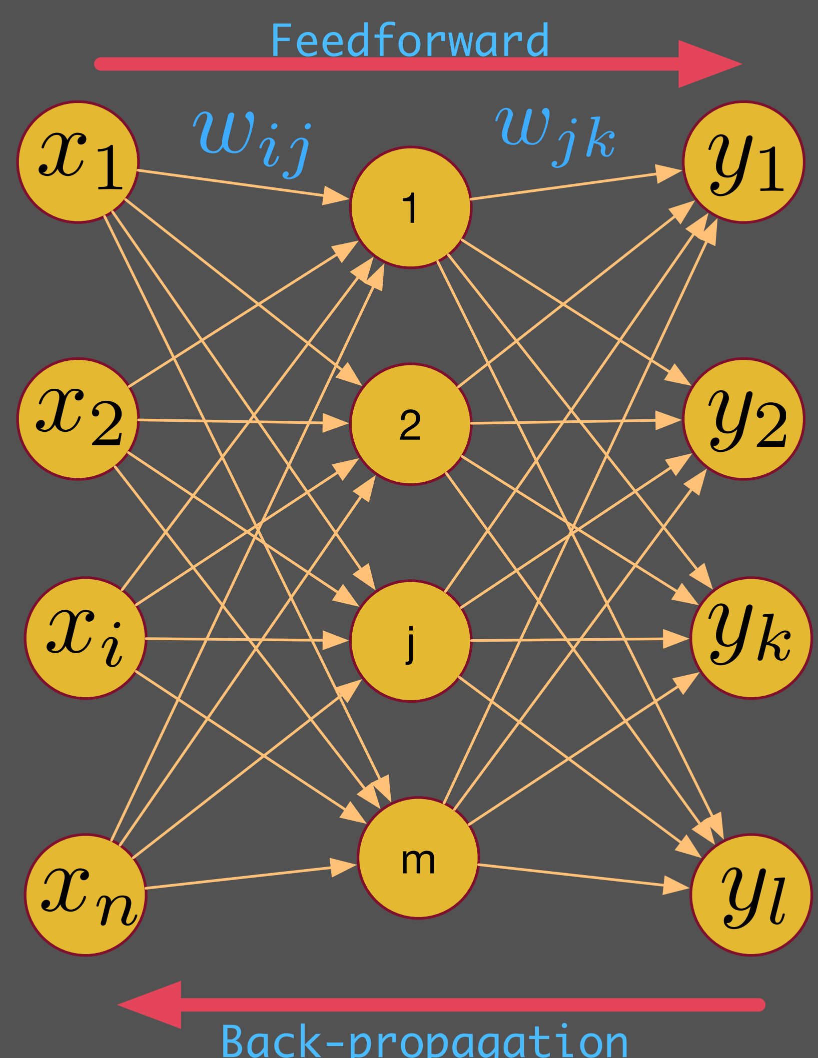
Back-propagation

- most popular learning rule for performing supervised learning tasks;
- No only used in Feedforward learning;
- propagates backward the error between the signal and the network output through the network;
- continuous, nonlinear, differentiable activation function
  - sigmoid functions, hyperbolic tangent;

Feedforward Networks

Back-propagation

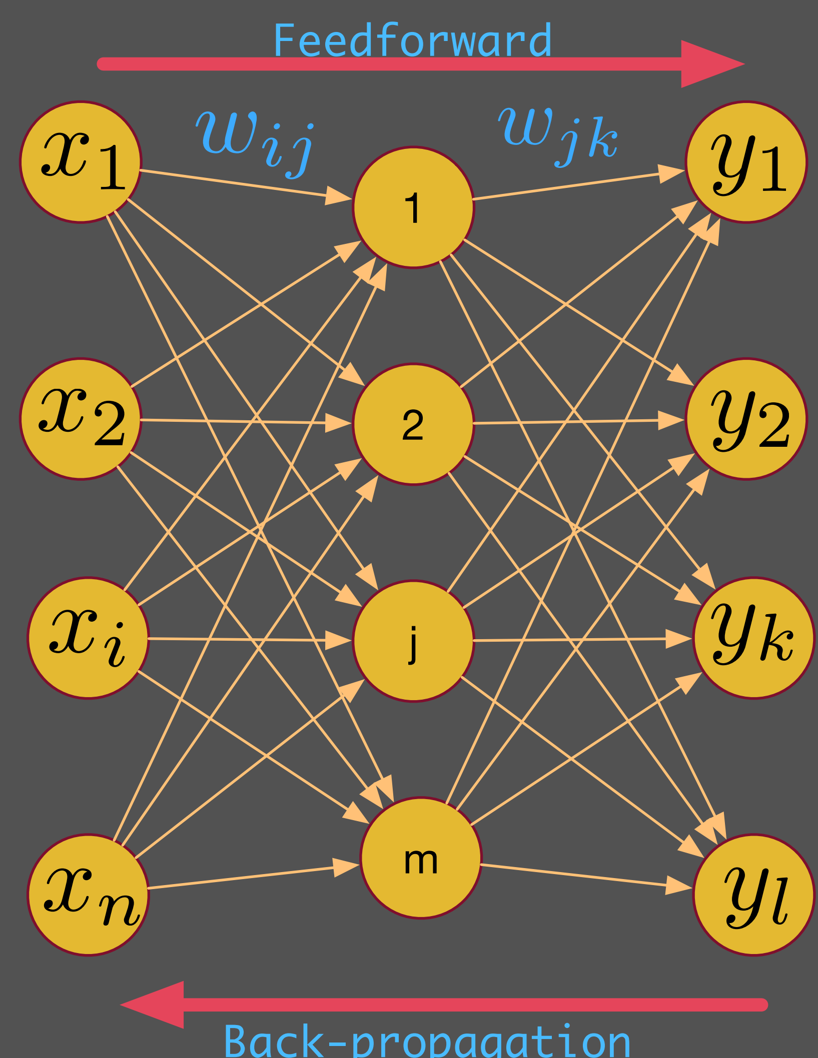
- Indexes i,j,k refer to neurons in input, hidden and output layers;



Feedforward Networks

Back-propagation

- Input signals flow from left to right and error signals from right to left;

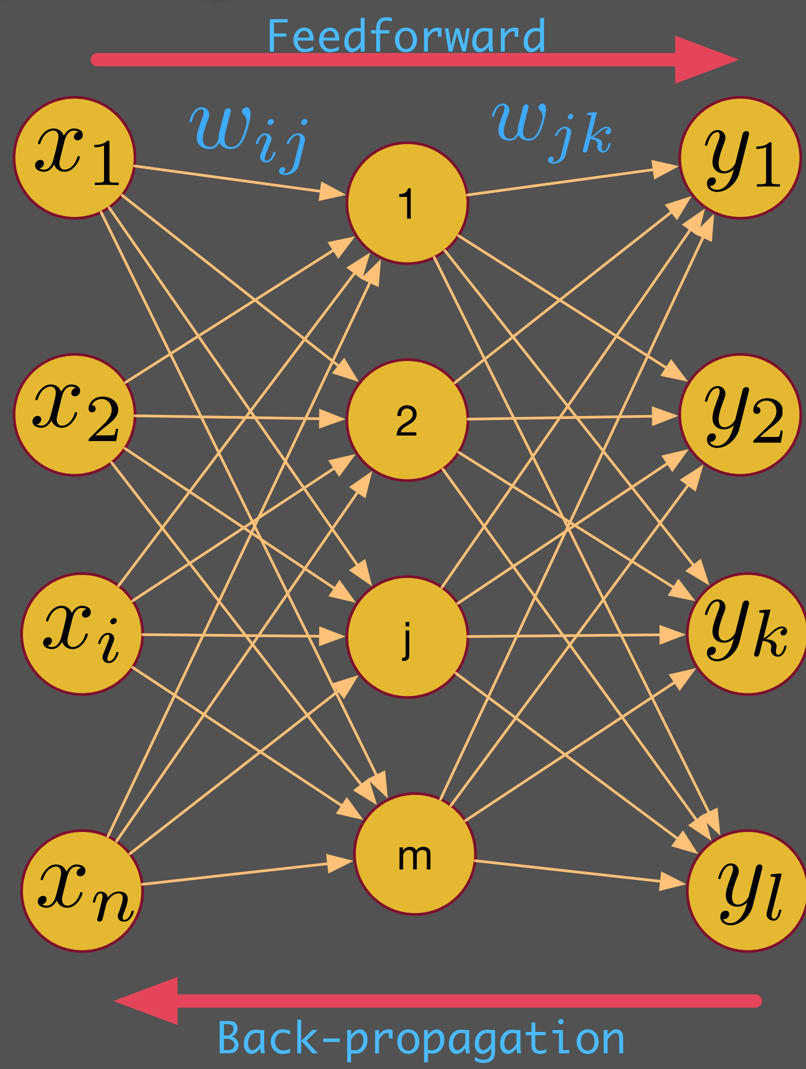


Feedforward Networks

Back-propagation

- w_{ij} represents the weight that connects the input neuron i and the neuron in the hidden layer j

- w_{jk} the weight between the neuron j in the hidden layer and the neuron k in the output layer



Feedforward Networks

Back-propagation

- objective function for optimization is defined as the MSE between the $\hat{y}_p$ and the desired output y_p :

$$e_{p,k} = \hat{y}_{p,k} - y_{p,k}$$

Feedforward Networks

Back-propagation

- first step is to correct the weight between j and k by minimizing the error;
  - error gradient, learning rate;

$$w_{p+1,jk} = w_{p,jk} + \Delta w_{p,jk}$$

$$\Delta w_{p,jk} = \alpha y_{p,j} \delta(\hat{y}_{p,k})$$

$$\Delta w_{p,ij} = \alpha x_{p,i} \delta(\hat{y}_{p,j})$$

Feedforward Networks

Back-propagation

- It gets uglier!

Neural Networks

Gets better

- Programming APIs:
  - PyTorch;
  - Theano;
  - TensorFlow;
  - …

Neural Networks

Gets better

```
def forwardprop(X, w_1, w_2):
    """ Forward-propagation """
    h    = T.nnet.sigmoid(T.dot(X, w_1))  # The \sigma function
    yhat = T.nnet.softmax(T.dot(h, w_2))  # The \varphi function
    return yhat


def backprop(cost, params, lr=0.01):
    """ Back-propagation """
    grads   = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates
```

Questions?