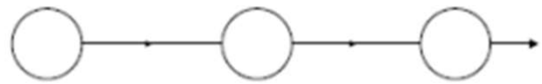
# Backpropagation algorithm

## *Back-propagation* algorithm, Summary



Initialize the weights to small random values.

repeat

- for each training example $<(x_1,\ldots x_n),t>$ Do
  - Input the instance $(x_1,\ldots,x_n)$ to the network and compute the outputs from each layer in the network : (**forward step**)
    - For output layer: $y_k=F(v_k)$, since $v_k=\Sigma w_{kh} y_h$
    - For hidden layer: $y_h=\Phi(v_h)$, since $v_h= \Sigma w_{hi} x_i$
  - Compute the errors signal $\delta_j$ in the output layer and propagate them to the hidden layer (**backward step**):
    - For each output unit k

      $\delta_k=(t_k-y_k)\ F'(v_k),$
    - For each hidden unit h

      $\delta_h= \Phi'(v_h)\Sigma_k\ w_{kh}\ \delta_k$
  - Update the weights in both layers according to: (**forward step**)

    $\Delta w_{kh} = \eta\ \delta_k\ y_h$ , $\Delta whi = \eta\ \delta_h\ x_i$
- end for loop

until overall error E becomes acceptably low

# Task2 description

1. Implement the **Back-Propagation learning algorithm** on a multi-layer neural networks, which can be able to classify a stream of input data to one of a set of predefined classes.

- Use the dry beans data in both your training and testing processes. (Each class has 50 samples: train NN with the first 30 non-repeated samples, and test it with the remaining 20 samples)

# Task2 description

## 2. After training

• Test the classifier with the remaining 20 samples of each selected classes and find confusion matrix and compute overall accuracy.

# Task2 description

**1. User Input:**

- Enter number of hidden layers
- Enter number of neurons in each hidden layer
- Enter learning rate (eta)
- Enter number of epochs (m)
- Add bias or not (Checkbox)
- Choose to use Sigmoid or Hyperbolic Tangent sigmoid as the activation function

**2. Initialization:**

- Number of features = 5
- Number of classes = 3
- Weights + Bias = small random numbers

**3. Classification:**

- Sample (single sample to be classified).

# Task2 description

**4. Workflow:**

- **Training Phase: (repeat the following $m$ epochs)**

  Assuming that we have $n$ training samples $\{sample_i : i = 1 \rightarrow n\}$

  - Fetch features ($x$) of $sample_i$, and its desired output ($d$)

  - ***Forward step*** **for input signal from input layer towards output layer**
    1. Calculate the net value ($v$) of each neuron,
    2. Calculate the output of each neuron using the selected activation function,

  - ***Backward step*** **for error signal from output layer towards input layer**

    Calculate error in each neuron

  - ***Update the weights***

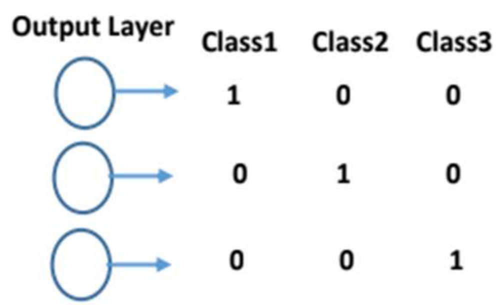
    ***new weights = old weights + eta * error * input***

# Task2 description

**4. Workflow:**

➢ **Testing Phase:**

1. Given a sample ***x***
2. **Forward step for input signal from input layer towards output layer**
   a. Calculate the net value (***v***) of each neuron,
   b. Calculate the output of each neuron using the selected activation function,
3. Reaching the output layer, calculate the actual output of each neuron. Set the maximum actual output to 1 and the other outputs to 0. (i.e. assign the sample ***x*** to the output neuron with the maximum actual output)
4. **Output: *y*** (Class ID).




➢ **Evaluation:** build the confusion matrix and overall accuracy.

# Task2 description

- Submit a report with your code. This report contains (with screenshots) the best accuracy you obtained using each activation function and what parameters were used for these results

- Example:

| Activation Function | Train Accuracy | Test Accuracy | LR | Epochs | #Layers | #HiddenNodes |
|---|---|---|---|---|---|---|
| Sigmoid | 70 | 50 | 0.01 | 1000 | 2 | 3,4 |
| Tanh | 60 | 40 | 0.001 | 5000 | 1 | 5 |