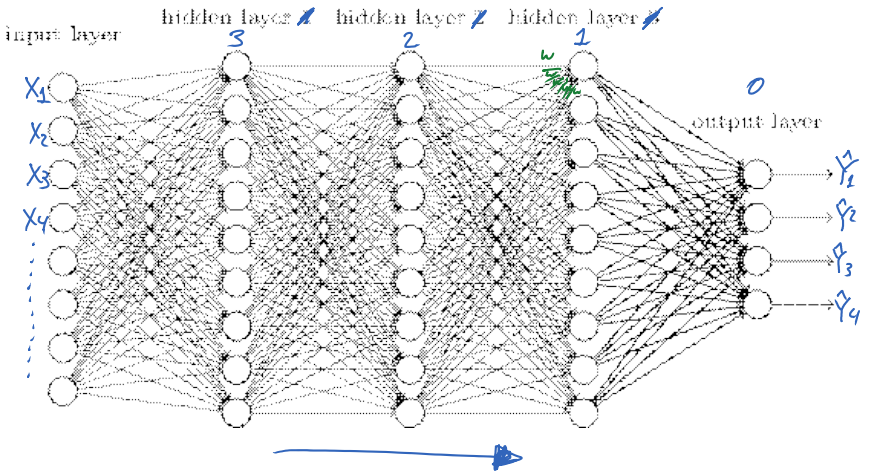
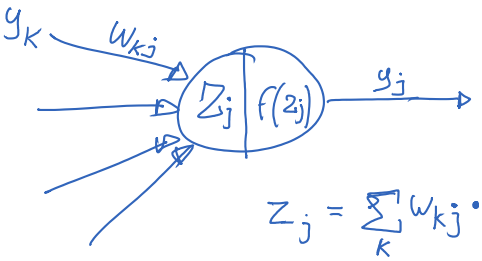
The Backpropagation Algorithm #2

Tuesday, October 8, 2019 2:03 PM

Attempt #2





$$z_j = \sum_k w_{kj} \cdot y_k$$

$$y_j = f(z_j) = sig(z_j)$$

$$z = w_1 a + w_2 b + w_3 c$$

$$\frac{\partial z}{\partial w_2} = b$$

To use gradient descent

$$\Delta w_{kj} = \eta \cdot \frac{\partial error}{\partial w_{kj}} \qquad \eta : \text{learning rate}$$

Case: Suppose j is the output layer

$$y_j = \hat{Y}_j$$

$$error = (Y_j - \hat{Y}_j)^2$$

$$\frac{\partial error}{\partial w_{kj}} ? = \frac{\partial error}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{kj}}$$

$$\delta_j \begin{cases} \frac{\partial error}{\partial y_j} = -2(Y_j - y_j) \\ \frac{\partial y_j}{\partial z_j} = y_j \cdot (1 - y_j) \end{cases}$$

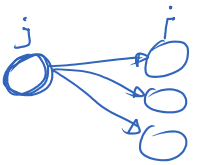
δ_j : error of our unit

$$\frac{\partial z_j}{\partial w_{kj}} = y_k$$

$$\frac{\partial error}{\partial w_{kj}} = -2(Y_j - y_j) \cdot y_j \cdot (1 - y_j) y_k = \delta_j y_k$$

Case: j is an inner layer.

$$\frac{\partial error}{\partial w_{kj}} = \frac{\partial error}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j} \cdot \frac{\partial z_j}{w_{kj}}$$



$$\delta_j \begin{cases} \frac{\partial error}{\partial y_j} = \sum_i \underbrace{\frac{\partial error}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_i}}_{\delta_i} \cdot \underbrace{\frac{\partial z_i}{\partial y_j}}_{w_{ji}} \\ \frac{\partial y_j}{\partial z_j} = y_j \cdot (1 - y_j) \end{cases}$$

$$\frac{\partial z_j}{w_{kj}} = y_k$$

$$\frac{\partial error}{\partial w_{kj}} = \left(\sum_i \delta_i \cdot w_{ji}\right) \cdot y_j \cdot (1 - y_j) \cdot y_k = \delta_j \cdot y_k$$

- Re-collect =

$$\Delta w_{kj} = \eta \cdot \delta_j y_k$$

η: learning rate

where

$$\delta_j = (Y_j - y_j) \cdot y_j \cdot (1 - y_j) \quad \text{if } j \text{ is an output layer}$$

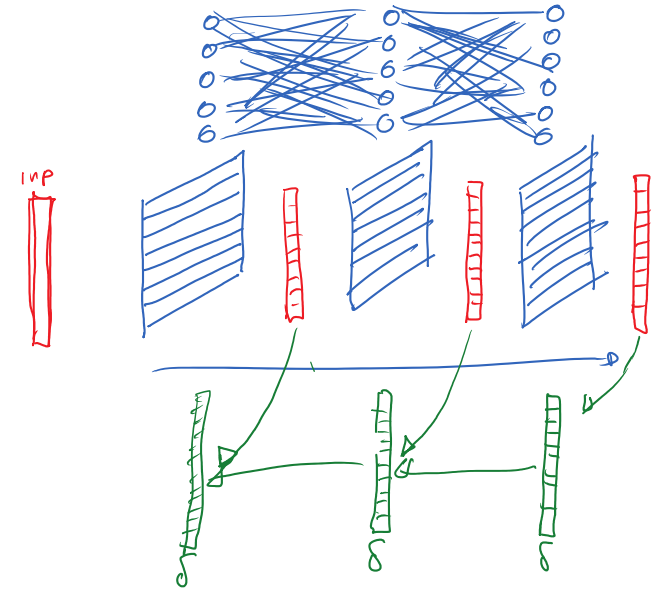
$$\delta_j = \left(\sum \delta_i w_{ji}\right) \cdot y_j \cdot (1 - y_j) \quad \text{if } j \text{ is an hidden layer.}$$

$$\delta_j = \left(\sum_i \delta_i w_{ji}\right) \cdot y_j \cdot (1 - y_j) \quad \text{if } j \text{ is an hidden layer.}$$

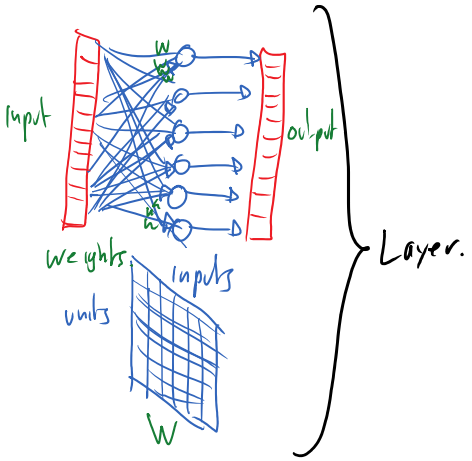
- Variation:

$$\Delta w_{kj} = \eta \cdot \delta_j y_k + \underbrace{\alpha \Delta w_{kj}}_{\text{momentum.}} \qquad \alpha \in [0..1)$$

IMPLEMENTATION



forward Propagation




```
CLASS Layer
  nu : number of units
  ni : number of inputs
  W[ ni+1 ][ nu ] : Unit's weights  // W[k][j] is weight applied to input k for unit j
  input[ ni+1 ] : input vector
  output[ nu ] : output vector
```

```
  PROCEDURE feedForward( inp[ ni ] )
      input := inp;
      input[ ni ] := 1;
      FOR every unit u in [ 0..nu ]
        output[ u ] := sig ( ∑k=0^ni W[ k ][ u ] * input[ k ] )

      RETURN output;


  deltaW[ ni+1 ][ nu ] : Unit's weights change // W[k][j] is weight from input k to unit j
  eta : Learning rate
  alpha : momentum constant
  delta_prev[ ni ] : error term of input

  PROCEDURE backProp( delta[ nu ] )
    FOR EACH input j in [0..ni]
        delta_prev[ j ] := ( ∑u=0^nu delta[ u ] * W[ j ][ u ] )
                           * input[ j ] * ( 1 - input[ j ] )

    FOR EACH input j in [0..ni]
      FOR EACH unit u in [0..nu]
        deltaW[ j ][ u ] := eta*delta[ u ]*output[ u ] + alpha*deltaW[ j ][ u ]
        W[ j ][ u ] := W[ j ][ u ] + deltaW[ j ][ u ]

    RETURN delta_prev;


PROCEDURE BackPropagation
  E  : Set of examples, each of the form ⟨ X, Y ⟩  where:
       X = ⟨ X_1, X_2, X_3, ..., X_nx ⟩
       Y = ⟨ Y_1, Y_2, Y_3, ..., Y_ny ⟩

  NN : A sequence of Layers
  Ycap[ ny ] : output of the Neural Network

  REPEAT
    FOR EACH example e in E
      values[] := e.X
      FOR EACH layer L in NN, from input to output
        values := L.feedForward( values )

      Ycap[] := values
      FOR EACH output j in [0..ny]
          delta[j] := ( e.Y_j - Ycap[y] ) * Ycap[y] * ( 1 - Ycap[y] )

      FOR EACH layer L in NN, from output to input
          delta := L.backProp( delta )

  UNTIL Termination
```

$\vec{O}_{ut} = \vec{w} \cdot \vec{I}^{T}$

δ_j

$\delta_j = \sum_i (\delta_i w_{ji})(y_j)(1-y_j)$

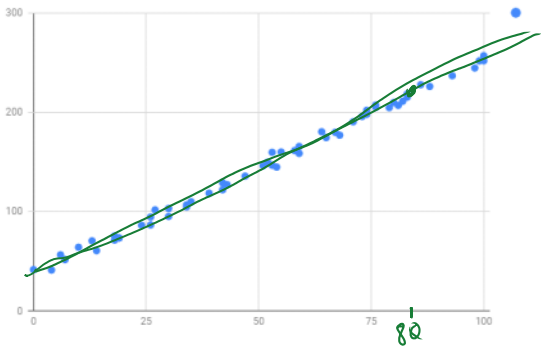
when using momentum

$\Delta w_{ju} = \eta \cdot \delta_u \cdot y_u + \alpha \Delta w_{ju}$

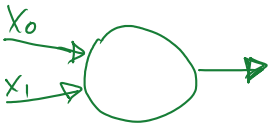
$\delta_j = (Y-\hat{Y})(\hat{Y})(1-\hat{Y})$

Stay Tuned !!!

Stay Tuned !!!



HW #1 Linear Regression.-



$y = \sum w_i x_i$

Preview: Convolutional NN