Loss function and Back propagation

Prof. (Dr.) Keyur Rana

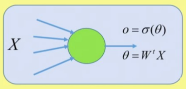
- Loss function / Error function
  - Sum of Square error (quadratic error) – to minimize

$$Div(Y,d) = \frac{1}{2}\|Y-d\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$$

Cross entropy loss – Two class problem

$o \Rightarrow$ likelihood that y *is* 1

$(1-o) \Rightarrow$ likelihood that y *is* 0



Likelihood that is to be maximized $\Rightarrow o^y(1-o)^{(1-y)}$

Loglikelihood $\Rightarrow y\log o + (1-y)\log(1-o)$
to maximized

Cross entropy loss – Two class problem

$$\text{Minimize} \Rightarrow C = -\frac{1}{N}\sum_{\forall X}\left[y\log o + (1-y)\log(1-o)\right]$$

$$\frac{\partial C}{\partial W_i} = -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial\sigma(\theta)}{\partial W_i}$$

$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial\sigma(\theta)}{\partial\theta}\cdot\frac{\partial\theta}{\partial W_i} \quad \text{(Chain rule is applied)}$$

o is σ(θ) = Sigmoidal (θ), here θ = $W^T X$

Cross entropy loss – Two class problem

Simplifying further…

$$\frac{\partial C}{\partial W_i} = -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial\sigma(\theta)}{\partial\theta}\cdot\frac{\partial\theta}{\partial W_i}$$

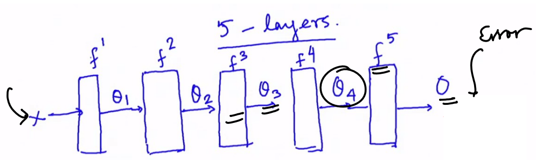
$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y}{\sigma(\theta)} - \frac{(1-y)}{1-\sigma(\theta)}\right]\frac{\partial\sigma(\theta)}{\partial\theta}\cdot\frac{\partial\theta}{\partial W_i}$$

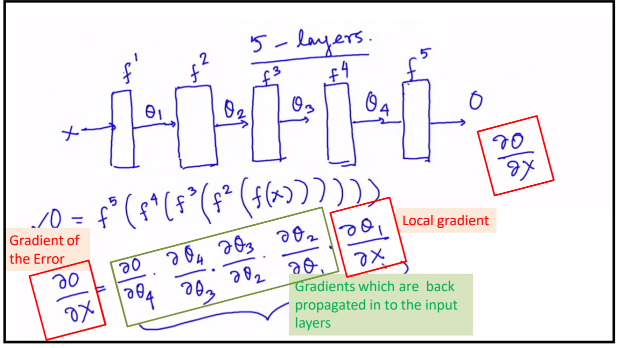
$$= -\frac{1}{N}\sum_{\forall X}\left[\frac{y-\sigma(\theta)}{\sigma(\theta)(1-\sigma(\theta))}\right]\underline{\sigma(\theta)(1-\sigma(\theta)}.\underline{x_i}$$

$$= \frac{1}{N}\sum_{\forall X} x_i(\sigma(\theta)-y) \qquad \boxed{= \frac{1}{N}\sum_{\forall X} x_i(o-y)}$$

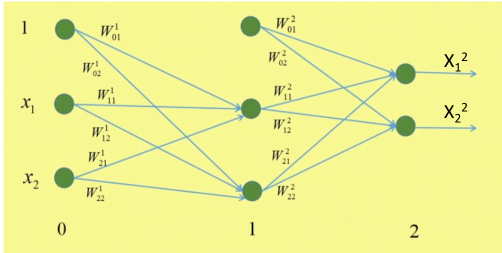
Derivative of Sigmodial function σ(x) is **σ(x)(1- σ(x))**

θ = $W^T X$
$\sigma(W_i X_i) = X_i$



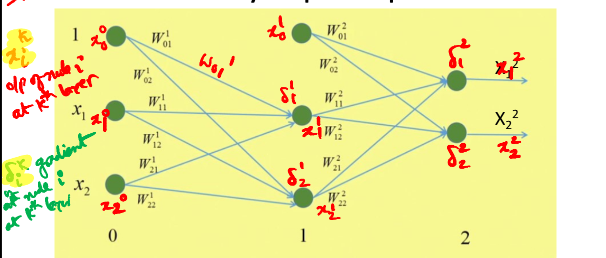


Multilayer perceptron



W_{ij}^k ⇒ weight from node i (from k-1 layer) to node j at k^{th} layer

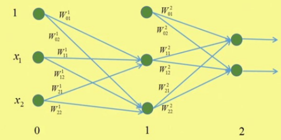
Multilayer perceptron



Multilayer perceptron

W^1_{01}	W^1_{11}	W^1_{21}
0.5	1.5	0.8
W^1_{02}	W^1_{12}	W^1_{22}
0.8	0.2	-1.6

W^2_{01}	W^2_{11}	W^2_{21}
0.9	-1.7	1.6
W^2_{02}	W^2_{12}	W^2_{22}
1.2	2.1	-0.2



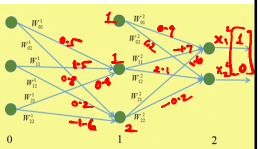
$$X = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix} \text{ from category 1 } \Rightarrow t = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Multilayer perceptron

W^1_{01}	W^1_{11}	W^1_{21}
0.5	1.5	0.8
W^1_{02}	W^1_{12}	W^1_{22}
0.8	0.2	-1.6

W^2_{01}	W^2_{11}	W^2_{21}
0.9	-1.7	1.6
W^2_{02}	W^2_{12}	W^2_{22}
1.2	2.1	-0.2

$\begin{bmatrix} 1 \\ 0.7 \\ 1.2 \end{bmatrix}$

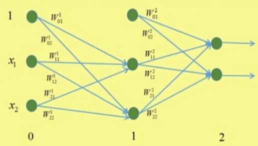


$$X = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix} \text{ from category 1 } \Rightarrow t = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Feed forward pass

W^1 $\quad x_i^0 \quad \theta_j^1 = \sum_i W_{ij}^1 x_i^0 \quad x_j^1 = \frac{1}{1+e^{-\theta_j^1}}$

$$\begin{bmatrix} 0.5 & 1.5 & 0.8 \\ 0.8 & 0.2 & -1.6 \end{bmatrix} \begin{bmatrix} 1 \\ 0.7 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 2.51 \\ -.98 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.92 \\ 0.27 \end{bmatrix}$$



W^2 $\quad x_i^1 \quad \theta_j^2 = \sum_i W_{ij}^2 x_i^1 \quad x_j^2 = \frac{1}{1+e^{-\theta_j^2}}$

$$\begin{bmatrix} 0.9 & -1.7 & 1.6 \\ 1.2 & 2.1 & -0.2 \end{bmatrix} \begin{bmatrix} 1 \\ 0.92 \\ 0.27 \end{bmatrix} = \begin{bmatrix} -0.232 \\ 3.078 \end{bmatrix} \Rightarrow \begin{bmatrix} 0.44 \\ 0.95 \end{bmatrix} \quad \boxed{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}$$

Mismatch!!
i.e. Error

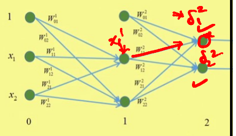
Back propagation – Output layer

$$E = \frac{1}{2}\sum_{j=1}^{2}\left(x_j^2 - t_j\right)^2 \qquad x_j^2 = \frac{1}{1+e^{-\theta_j^2}} \qquad \theta_j^2 = \sum_{i=0}^{2} W_{ij}^2 x_i^1$$

$$\frac{\partial E}{\partial W_{ij}^2} = \frac{\partial E}{\partial x_j^2}\cdot\frac{\partial x_j^2}{\partial \theta_j^2}\cdot\frac{\partial \theta_j^2}{\partial W_{ij}^2} = (x_j^2 - t_j)x_j^2(1-x_j^2)x_i^1$$

We set $\boxed{\delta_j^2 = x_j^2(1-x_j^2)(x_j^2 - t_j)} \Rightarrow \frac{\partial E}{\partial W_{ij}^2} = \delta_j^2 x_i^1$

$$\boxed{W_{ij}^2 \leftarrow W_{ij}^2 - \eta \frac{\partial E}{\partial W_{ij}^2}}$$



Back propagation – Output layer

$$\boxed{\delta_j^2 = x_j^2(1-x_j^2)(x_j^2 - t_j)}$$

$$\delta_1^2 = x_1^2(1-x_1^2)(x_1^2 - t_1) = 0.44*(1-0.44)*(0.44-1) = -0.138$$

$$\Rightarrow \frac{\partial E}{\partial W_{11}^2} = \delta_1^2 x_1^1 = -0.126$$

$$\delta_2^2 = x_2^2(1-x_2^2)(x_2^2 - t_2) = 0.95*(1-0.95)*(0.95-0) = 0.045$$

$$\Rightarrow \frac{\partial E}{\partial W_{12}^2} = \delta_2^2 x_1^1 = 0.04$$

Observe the updated weights

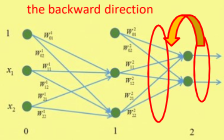
$$\boxed{W_{11}^2 \leftarrow W_{11}^2 + \eta * 0.126} \qquad \boxed{W_{12}^2 \leftarrow W_{12}^2 - \eta * 0.04}$$

Back propagation – Output layer

$$\frac{\partial E}{\partial W_{21}^2} = \delta_1^2 x_2^1 = -0.037 \qquad \frac{\partial E}{\partial W_{22}^2} = \delta_2^2 x_2^1 = 0.012$$

$$\frac{\partial E}{\partial W_{01}^2} = \delta_1^2 x_0^1 = -0.138 \qquad \frac{\partial E}{\partial W_{02}^2} = \delta_2^2 x_0^1 = 0.045$$

Error from output is propagated in the backward direction



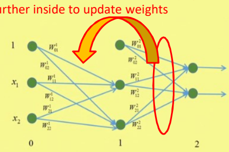
Weights in the output layer are updated

Back propagation – Output layer

These errors has to be further propagated further inside to update weights

$$\frac{\partial E}{\partial W_{21}^2} = \delta_1^2 x_2^1 = -0.037 \qquad \frac{\partial E}{\partial W_{22}^2} = \delta_2^2 x_2^1 = 0.012$$

$$\frac{\partial E}{\partial W_{01}^2} = \delta_1^2 x_0^1 = -0.138 \qquad \frac{\partial E}{\partial W_{02}^2} = \delta_2^2 x_0^1 = 0.045$$



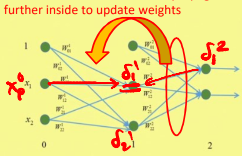
Weights in the output layer are updated

Back propagation – Output layer

These errors has to be further propagated further inside to update weights

$$\frac{\partial E}{\partial W_{21}^2} = \delta_1^2 x_2^1 = -0.037 \qquad \frac{\partial E}{\partial W_{22}^2} = \delta_2^2 x_2^1 = 0.012$$

$$\frac{\partial E}{\partial W_{01}^2} = \delta_1^2 x_0^1 = -0.138 \qquad \frac{\partial E}{\partial W_{02}^2} = \delta_2^2 x_0^1 = 0.045$$



Weights in the output layer are updated

$$\frac{\partial E}{\partial w_{11}} = \delta_1^1 x_1^0$$

$w_{11} \leftarrow$

Back propagated Error term

$$\delta_i^k = x_i^k(1-x_i^k)\sum_{j=1}^{M_{k+1}} \delta_j^{k+1} W_{ij}^{k+1}$$

- **δ_i^k** : Back propagated Error Term at layer k for the i^{th} node
- All the nodes to which i^{th} node has fed the input, **accumulate** the error corresponding to the weights and Multiply with the local derivative *xi(1-xi)*

Back propagation – Hidden layer

Local derivative of the output of the ith node

We set $\boxed{\delta_i^k} = \boxed{x_i^k(1-x_i^k)}\sum_{j=1}^{M_{k+1}} \delta_j^{k+1} W_{ij}^{k+1} \Rightarrow \delta_i^1 = x_i^1(1-x_i^1)\sum_{j=1}^{2}\delta_j^2 W_{ij}^2 \Rightarrow \frac{\partial E}{\partial W_{ij}^k} = \delta_j^k x_i^{k-1}$

Propagated error

$$\begin{aligned}\delta_1^1 &= x_1^1(1-x_1^1)\left[\delta_1^2 * W_{11}^2 + \delta_2^2 W_{12}^2\right] \\ &= 0.92*(1-0.92)[(-0.137)*(-1.7)+0.045*2.1] \\ &= 0.024\end{aligned}$$

$$\begin{aligned}\delta_2^1 &= x_2^1(1-x_2^1)\left[\delta_1^2 * W_{21}^2 + \delta_2^2 W_{22}^2\right] \\ &= 0.27*(1-0.27)[(-0.137)*0.8+0.045*(-0.2)] \\ &= -0.02\end{aligned}$$

Back propagation – Hidden layer

$$\frac{\partial E}{\partial W_{11}^1} = \delta_1^1 * x_1^0 = 0.024*0.7 = 0.017$$

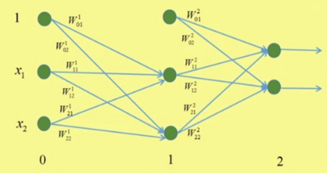
$$\frac{\partial E}{\partial W_{12}^1} = \delta_2^1 * x_1^0 = -0.02*0.7 = -0.014$$

$$\frac{\partial E}{\partial W_{21}^1} = \delta_1^1 * x_2^0 = 0.024*1.2 = 0.0288$$

$$\frac{\partial E}{\partial W_{22}^1} = \delta_2^1 * x_2^0 = -0.02*1.2 = -0.024$$

$$\frac{\partial E}{\partial W_{01}^1} = \delta_1^1 * x_0^0 = 0.024*1 = 0.024$$

$$\frac{\partial E}{\partial W_{02}^1} = \delta_2^1 * x_0^0 = -0.02*1 = -0.02$$



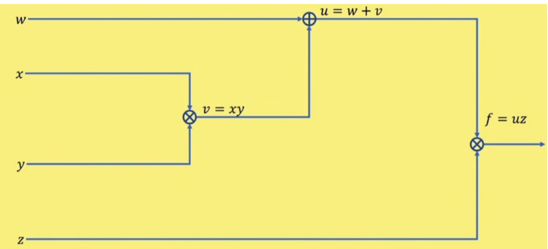
$$W_{ij}^1 \leftarrow W_{ij}^1 - \eta \frac{\partial E}{\partial W_{ij}^1}$$

Back propagation

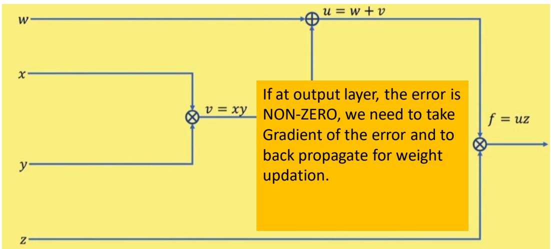
- Applicable to any number of layers
- Highly parallelism possible

Back propagation learning at the Node level

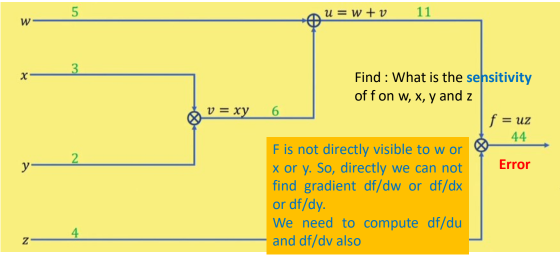
Back propagation



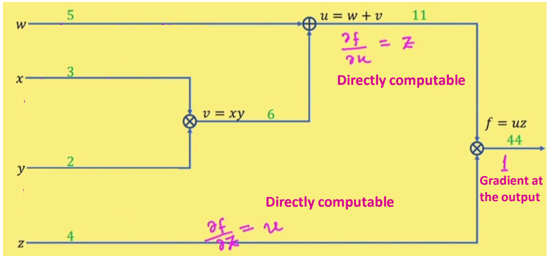
Back propagation



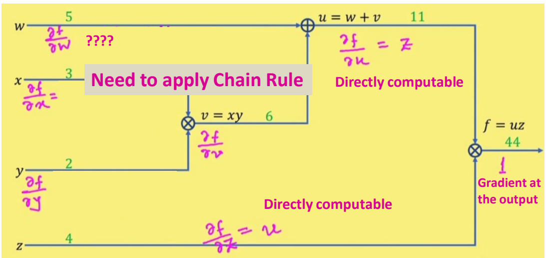
Back propagation



Back propagation

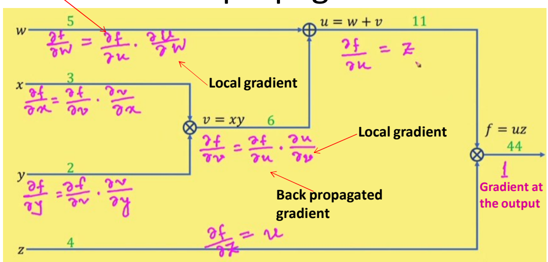


Back propagation



Back propagated gradient

Back propagation



Back propagation

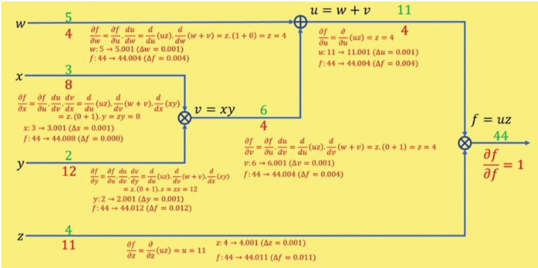
```
# Set Input
  w=5; x=3; y=2; z=4

# Forward Pass          # Backward Pass
  v = x*y                 dfdu = z
  u = w+v                 dfdz = u
  f = u+z                 dfdw = 1*dfdu   # dudw = 1
                          dfdv = 1*dfdu   # dudv = 1
                          dfdx = y*dfdv   # dvdx = y
                          dfdy = x*dfdv   # dvdy = x
```


Back propagation

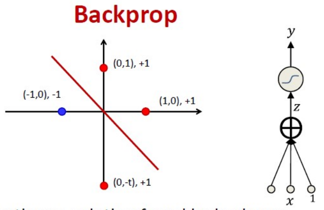
```
# Set Input
  w=5; x=3; y=2; z=4

# Forward Pass          # Backward Pass
  v = x*y                 dfdu = z
  u = w+v                 dfdz = u
  f = u+z                 dfdw = 1*dfdu   # dudw = 1
                          dfdv = 1*dfdu   # dudv = 1
                          dfdx = y*dfdv   # dvdx = y
                          dfdy = x*dfdv   # dvdy = x
```

sensitivity of f on w, x, y and z

Back propagation



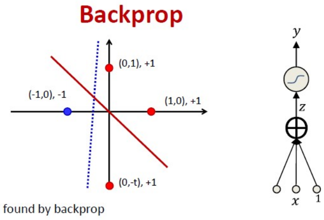
Backprop



- Local optimum solution found by backprop
- Does not separate the points *even though the points are linearly separable!*

21

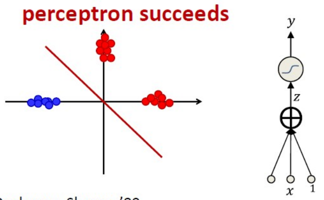
Backprop



- Solution found by backprop
- Does not separate the points *even though the points are linearly separable!*
- Compare to the perceptron: *Backpropagation fails to separate where the perceptron succeeds*

22

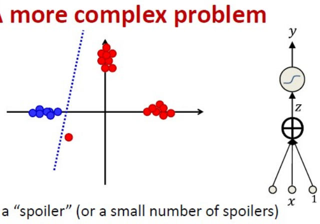
Backprop fails to separate where perceptron succeeds



- Brady, Raghavan, Slawny, ’89
- *Several* linearly separable training examples
- Simple setup: **both backprop and perceptron algorithms find solutions**

23

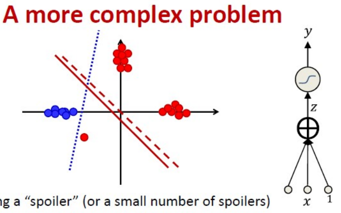
A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,

24

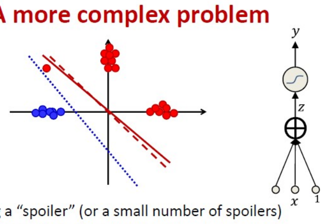
A more complex problem



- Adding a “spoiler” (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - Backprop does not find a separator
    - A single additional input does not change the loss function significantly
      - **Assuming weights are constrained to be bounded**

25

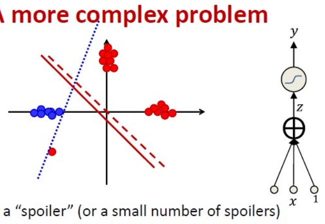
A more complex problem



- Adding a "spoiler" (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded w, backprop does not find a separator
    - A single additional input does not change the loss function significantly

26

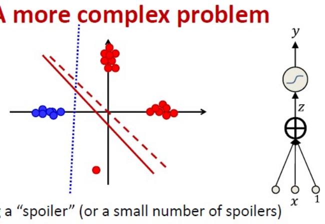
A more complex problem



- Adding a "spoiler" (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded w, backprop does not find a separator
    - A single additional input does not change the loss function significantly

27

A more complex problem



- Adding a "spoiler" (or a small number of spoilers)
  - Perceptron finds the linear separator,
  - For bounded w, backprop does not find a separator
    - A single additional input does not change the loss function significantly

28

So what is happening here?

- The perceptron may change greatly upon adding just a single new training instance
  - But it fits the training data well
  - The perceptron rule has *low bias*
    - Makes no errors if possible
  - But high variance
    - Swings wildly in response to small changes to input
- Backprop is minimally changed by new training instances
  - Prefers consistency over perfection
  - It is a *low-variance* estimator, at the potential cost of bias

29

Backpropagation: Finding the separator

- Backpropagation will often not find a separating solution ***even though the solution is within the class of functions learnable by the network***

- This is because the separating solution is not a feasible optimum for the loss function
- One resulting benefit is that a backprop-trained neural network classifier has lower variance than an optimal classifier for the training data

32