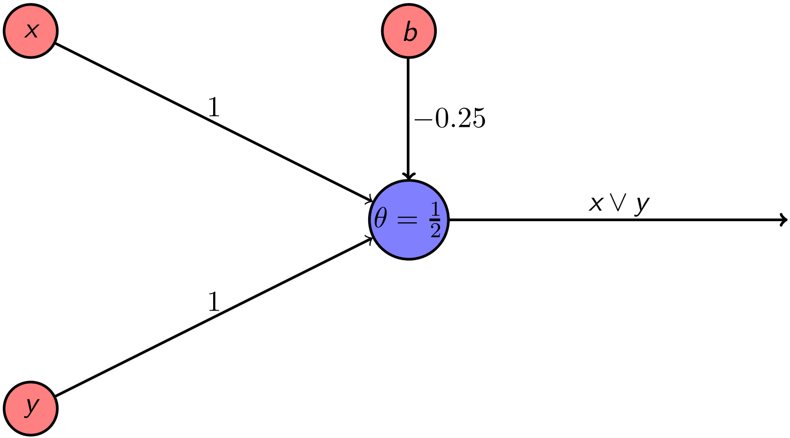
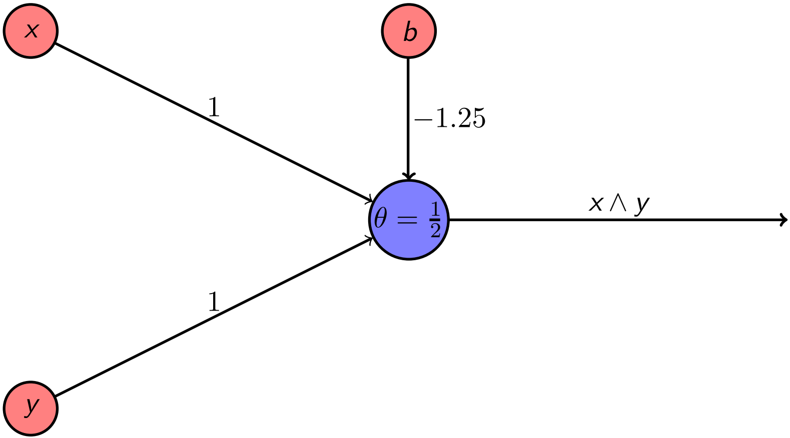
Lecture 19 contd: Neural Network Training using Backpropagation, Convolutional And Recurrent Neural Networks

Instructor: Prof. Ganesh Ramakrishnan

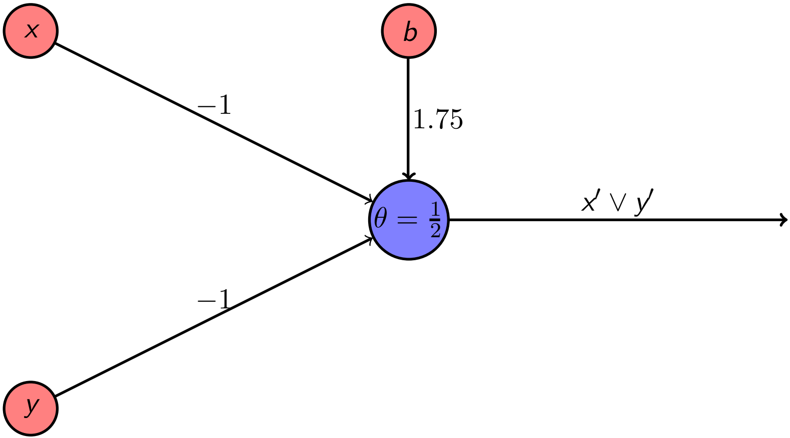
OR using perceptron



AND using perceptron

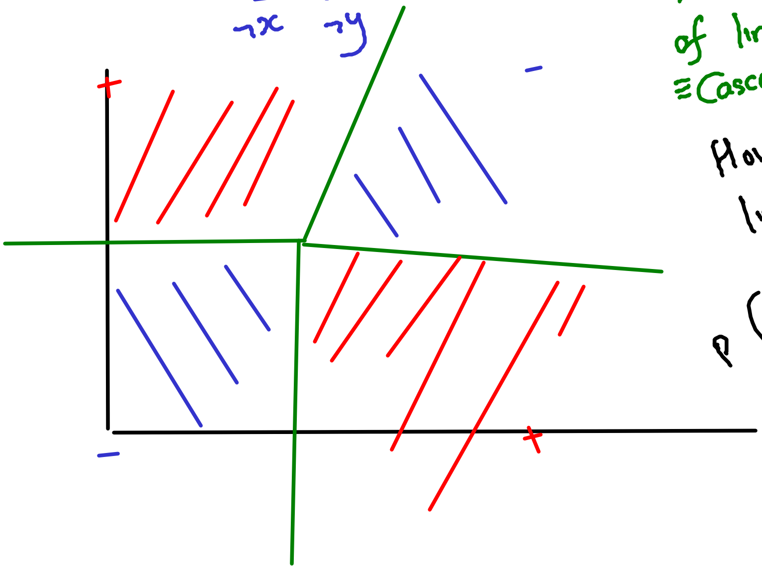


x'Vy' using perceptron



How about XOR $((x' \vee y') \wedge (x \vee y))$?

$\neg x \quad \neg y$

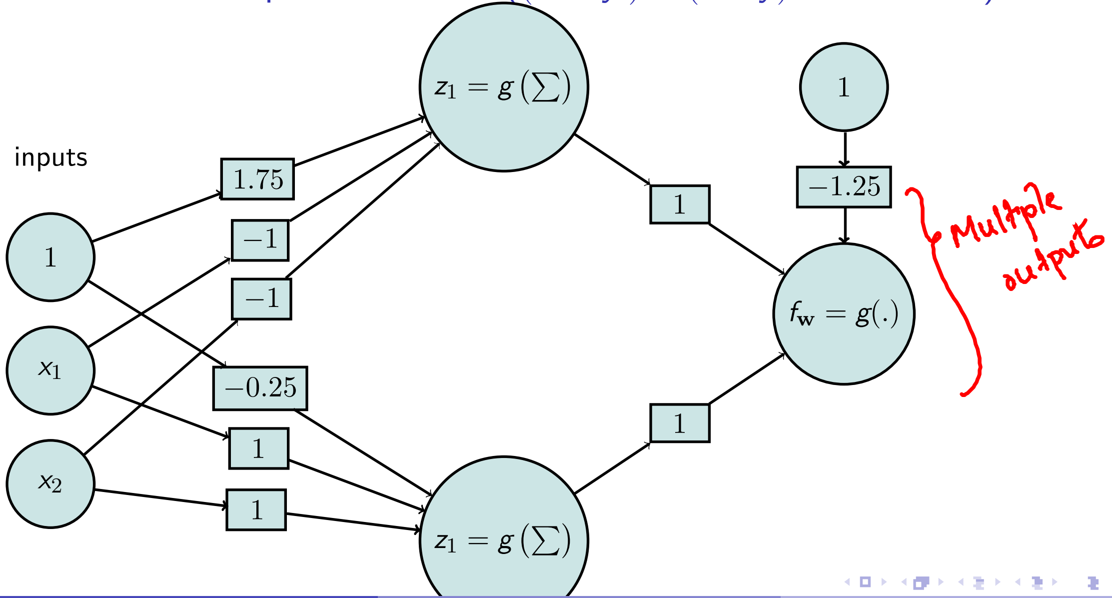


Functional composition of linear seperators
≡ Cascade of perceptrons

How about composing linear seperators?

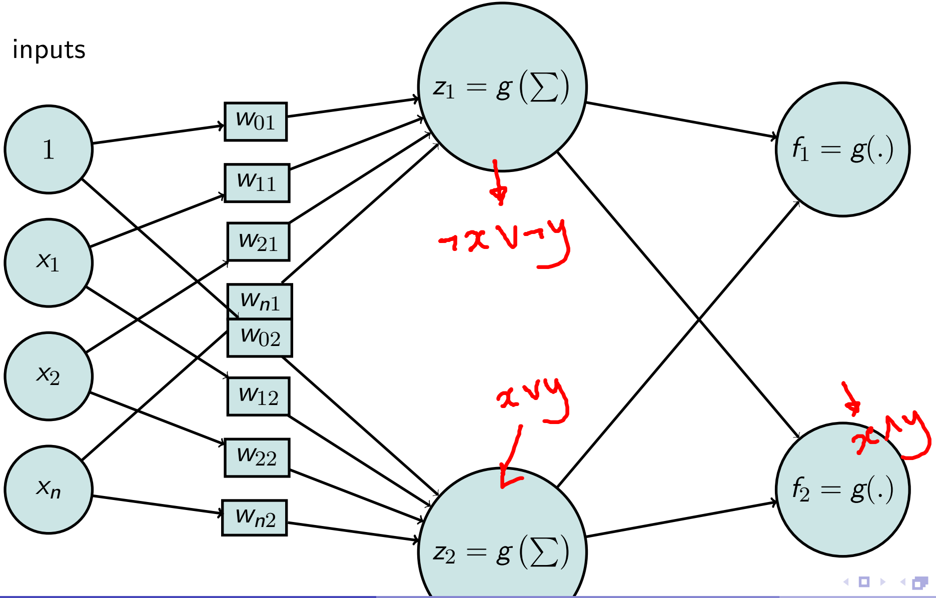
$P\left(P(\bar{x} \vee \bar{y}) \wedge P(x \vee y)\right)$

Cascade of Perceptrons for XOR($(x' \vee y') \wedge (x \vee y)$ with $\theta = 0$)



Feed-forward Neural Nets

inputs



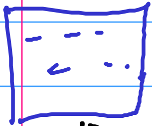
Training a Neural Network

STEP 0: Pick a network architecture → should be correctly suggested either by data or by expert

- Number of input units: Dimension of features $\phi\left(\mathbf{x}^{(i)}\right)$.
- Number of output units: Number of classes. → $y_1, y_2 \ldots y_k \in \{0,1\}$ .. Designed for multi labeling
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same number of hidden units in every layer.
- Number of hidden units in each layer a constant factor (3 or 4) of dimension of x.
  Rule of thumb
- We will use
  - the smooth sigmoidal function $g(s) = \frac{1}{1+e^{-s}}$: **We have now learnt how to train a single node sigmoidal (LR) neural network**
  - instead of the non-smooth step function $g(s) = 1$ if $s \in [\theta, \infty)$ and $g(s) = 0$ otherwise.

$\underbrace{1}_{1}$ $\underbrace{2}_{2}$ } Multiple class, Single label (per example)

[20 NewsGroups Dataset]



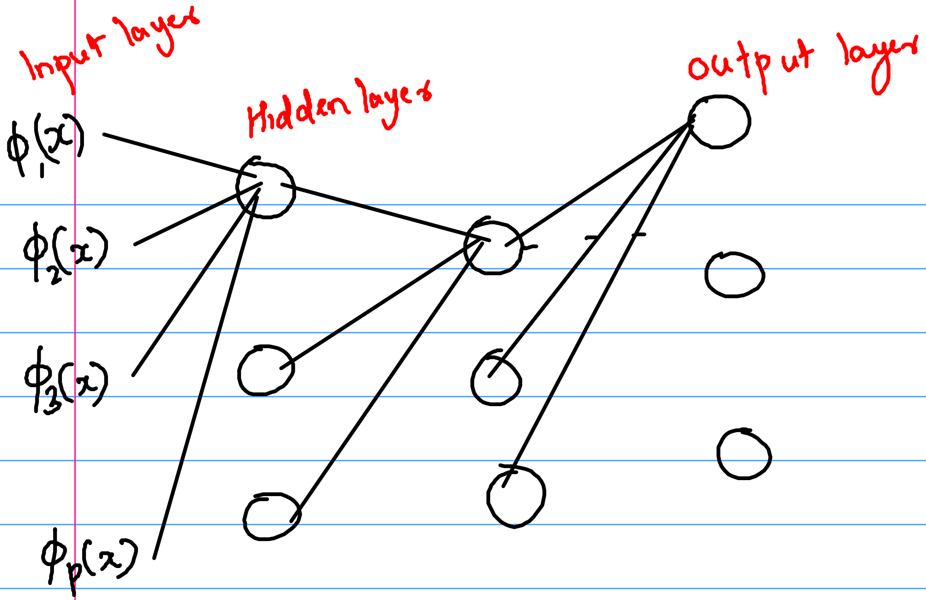
ML Text mining

M2 Car driving

} Multiple class, Multilabel

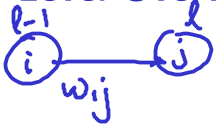
labels 1,2 ← { — — — } → set of instances ← { — — — } → labels 3,4

} Multi instance Multi label (MIML)



Input layer
Hidden layer
Output layer
$\phi_1(x)$
$\phi_2(x)$
$\phi_3(x)$
$\phi_p(x)$

High Level Overview of Backpropagation Algorithm for Training NN



Index into layer

1. Randomly initialize weights w^l_{ij} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_w(\mathbf{x})$ for any $x \in \mathcal{D}$. [eg: For XOR]
3. Execute **backpropagation**
   1. by computing partial derivatives $\frac{\partial}{\partial w^{(l)}_{ij}} E(w)$ for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
   2. and using gradient descent to try to minimize (non-convex) $E(w)$ as a function of parameters $\mathbf{w}$.
4. Verify that the cost function $E(w)$ has indeed reduced, else resort to some random perturbation of weights $\mathbf{w}$.

later

Intuition for Backpropagation

$f(x, y, z) = (w_x x + w_y y) w_q + w_z z, \; q = w_x x + w_y y$

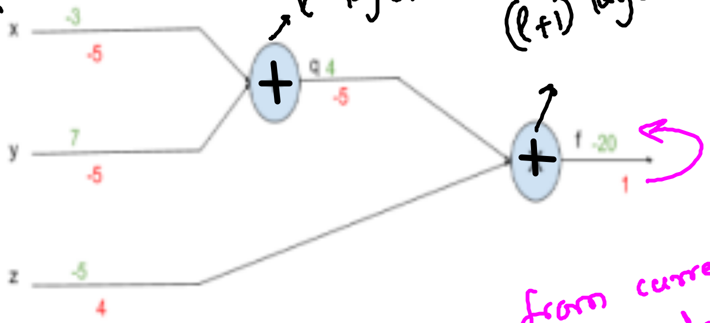
At input (-3,7,-5):- from (l-1) layer

(gradient of f w.r.t. v shown in red where $v \in \{w_x, w_y, w_z, w_q\}$)

(l-1)th layer

lth layer

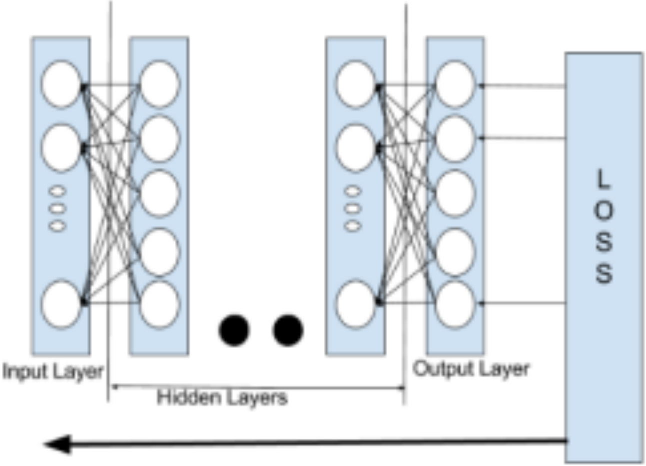
(l+1) layer



from current (1) to desired (-20)

Should we decrease w_y or should we increase w_x to decrease f?

Intuition for Backpropagation



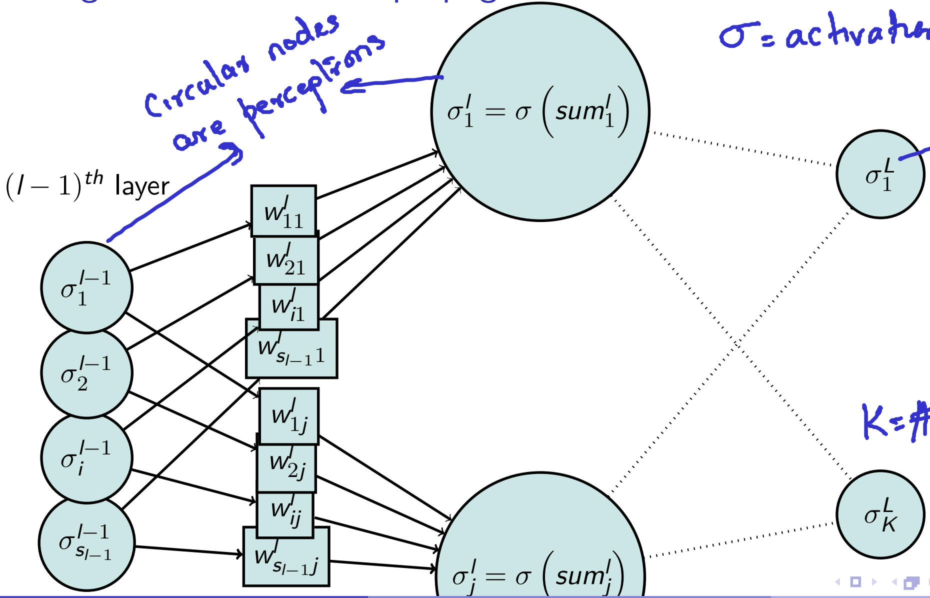
Iteratively update the weights along the direction where Loss decreases.

Setting Notation for Backpropagation

S_l = # of perceptrons in l^{th} layer

σ = activation fn (sigmoid)

Circular nodes are perceptrons



L = final or output layer.

K = # of output nodes.

Gradient Computation

- The Neural Network objective to be minimized:

output at final (L^{th}) layer

$$E(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)\right] \tag{1}$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2$$

Summation over all examples

Summation over all outputs

Regularizer computed on weights across all layers.

Cross entropy loss component computed only in terms of final output layer & $\underbrace{\text{input nodes}}_{(x^{(i)},\, y^{(i)})}$

Expand on: $w_{ij}^{l,(k+1)} = w_{ij}^{l,(k)} - \left(\frac{\partial E}{\partial w_{ij}^{l}}\right)_{w^{(k)}}$

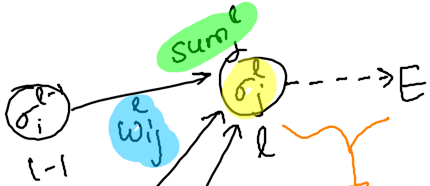
Gradient Computation

- The Neural Network objective to be minimized:

In $\frac{\partial E}{\partial \sigma_j^l}$ if $l = L$ then directly differentiate

$$E(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_k^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_k^L\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \quad (1)$$

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial \sigma_j^l}\frac{\partial \sigma_j^l}{\partial sum_j^l}\frac{\partial sum_j^l}{\partial w_{ij}^l}$ → chain rule
- $\frac{\partial sum_j^l}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^l}\left(\sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}\right) = \sigma_i^{l-1}$
- $\frac{\partial \sigma_j^l}{\partial sum_j^l} = \left(\frac{1}{1+e^{-sum_i^l}}\right)\left(1 - \frac{1}{1+e^{-sum_i^l}}\right)$



Many other layers.

Computing $\frac{\partial E}{\partial \sigma_j^l}$ requires Back Propagation

$\sigma_j \rightarrow \sigma_p^{l+1}$

- For a single example $(\mathbf{x}, y)$:

$$-\left[\sum_{k=1}^{K} y_k \log\left(\sigma_k^L(\mathbf{x})\right) + (1-y_k)\log\left(1-\sigma_k^L(\mathbf{x})\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \tag{2}$$

- $\frac{\partial E}{\partial \sigma_j^l} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial sum_p^{l+1}} \frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial \sigma_j^{l+1}} \frac{\partial \sigma_j^{l+1}}{\partial sum_p^{l+1}} w_{jp}^{l+1}$ since $\frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = w_{jp}^{l+1}$
- $\underbrace{\frac{\partial E}{\partial \sigma_j^L} = -\frac{y_j}{\sigma_j^L} - \frac{1-y_j}{1-\sigma_j^L}}$

Recall from logistic update.