Lecture 21: Neural Networks, Back-propagation, etc

Instructor: Prof. Ganesh Ramakrishnan

Non-linear perceptron?

- Kernelized perceptron: $f(x) = sign\left(\sum_i \alpha_i y_i K(x, x_i) + b\right)$
  - INITIALIZE: α=zeroes()
  - REPEAT: for $< x_i, y_i >$
    - If $sign\left(\sum_j \alpha_j y_j K(x_j, x_j) + b\right) \neq y_i$
    - then, $\alpha_i = \alpha_i + 1$
    - endif
- Neural Networks: Cascade of layers of perceptrons giving you non-linearity
  - $sign\left((w^*)^T \phi(x)\right)$ replaced by $g\left((w^*)^T \phi(x)\right)$ where $g(s)$ is a
    1. step function: $g(s) = 1$ if $s \in [\theta, \infty)$ and $g(s) = 0$ otherwise OR
    2. sigmoid function: $g(s) = \frac{1}{1+e^{-s}}$ with possible thresholding using some θ (such as $\frac{1}{2}$).
    3. Rectified Linear Unit (ReLU): $g(s) = max(0, s)$: A most popular activation function
    4. Softplus: $g(s) = \ln(1 + e^s)$

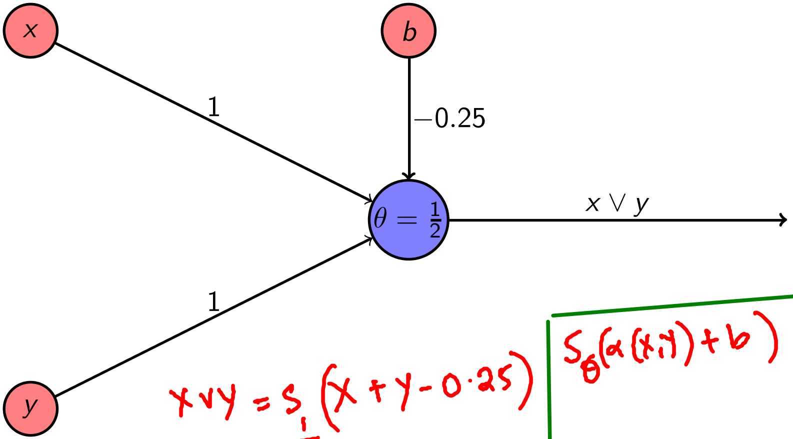
  Options 2, 3 and 4 have the thresholding step deferred. Threshold changes as bias is changed.

Recall: Measure for Linear non-separability?

- **Aspect 1: Number of functions that can be represented** Recall from Tutorial 6, Problem 1: Given n boolean variables how many of 2^{2^n} boolean functions can be represented by a perceptron? Ans: For 2 it is 14, for 3 it is 104, for 4 it is 1882
- **Aspect 2: Cardinality of largest set of points that can be shattered** VC (VapnikChervonenkis) dimension $\Rightarrow$ A measure of the richness of a space of functions that can be learned by a statistical classification algorithm.
  - A classification function $f(\mathbf{w})$ is said to shatter a set of data points $(x_1, x_2, \ldots, x_n)$ if, for all assignments of labels to those points, there exists a $\mathbf{w}$ such that $f(\mathbf{w})$ makes no errors when evaluating that set of data points.
  - Cardinality of the largest set of points that $f(\mathbf{w})$ can **shatter** is its VC-dimension.
  - Eg: For f as a threshold interval, VC dimension $= 1$
  - Eg: For f as an interval classifier, VC dimension $= 2$
  - Eg: For f as linear classifier (in 2 dimensions), VC dimension $= 3$
  - Eg: For f as linear classifier (in $\Re^n$), VC dimension $=$ n$+$ 1
  - Eg: For f as a neural network with sigmoid function, V nodes and E edges, then the VC dimension is at least $\Omega(|E|^2)$ and at most $O(|E|^2 \cdot |V|^2)$

OR using (step) perceptron

(For AND, $b = -1.25$)

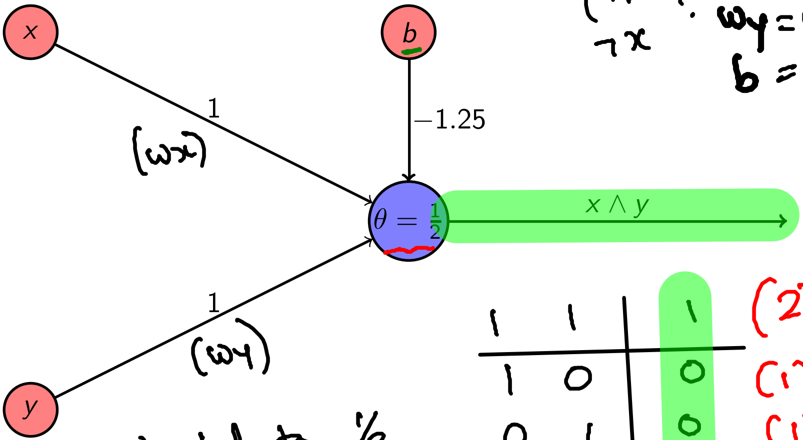


$x \vee y = S_{\frac{1}{2}}(x + y - 0.25)$

$S_\theta(\alpha(x,y) + b) =$ step fn $= 1$ if $\alpha + b > \theta$

$= 0$ o/w.

AND using (step) perceptron

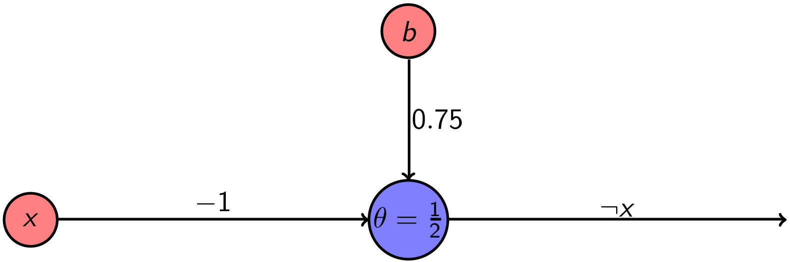


(NOT? $\neg x$. $w_x = -1$, $w_y = 0$, $b = 0.75$)

1	1	1	$(2) + b > 1/2$
1	0	0	$(1) + b < 1/2$
0	1	0	$(1) + b < 1/2$
0	0	0	$(0) + b < 1/2$

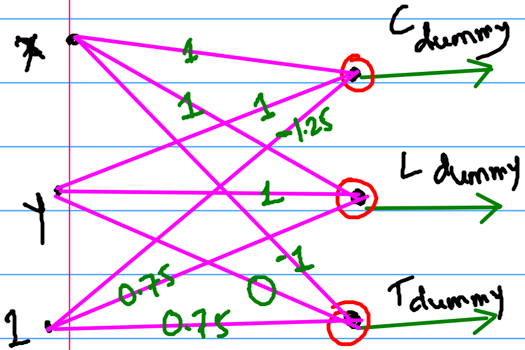
Note: θ restricted to $1/2$ to retain probabilistic interpretation

NOT using perceptron



$(x \wedge y)$ $\quad$ $(x \vee y)$ $\quad$ $(\neg x)$

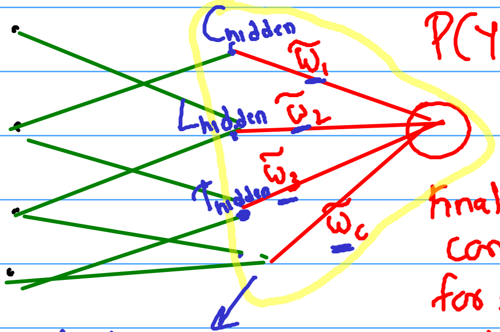
C_{dummy} $\quad$ L_{dummy} $\quad$ T_{dummy}



⊙ : Step fn with $\theta = 1/2$

(or sigmoid fn with $\theta = 1/2$)

In practice when only one of K classes should be predicted, use a softmax generalization of sigmoid as last layer

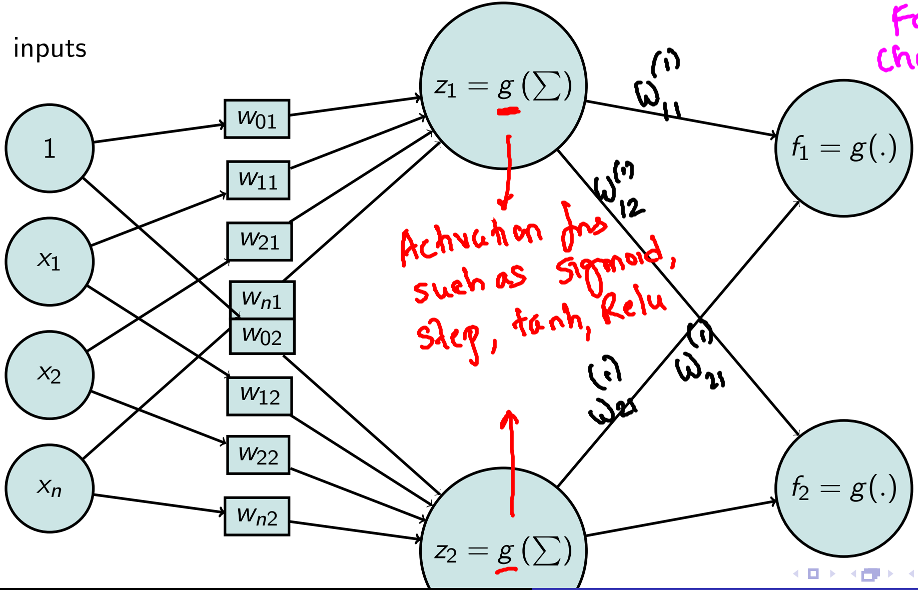


$$P(Y=c|x) = \frac{e^{-\tilde{w}_c^T \phi(x)}}{\sum_{c=1}^{K} e^{-\tilde{w}_c^T \phi(x)}}$$

final softmax consolidation for single label, multiclass classification

At penultimate layer, you get multilabel multiclass classification

Feed-forward Neural Nets



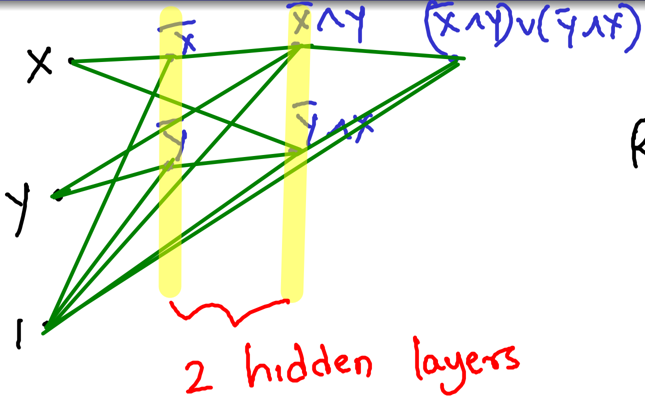
For Tut 7, prob 1
char recognition
3 o/p nodes

C

L

T

Eg: Feed-forward Neural Net for XOR ($\theta = 0$)



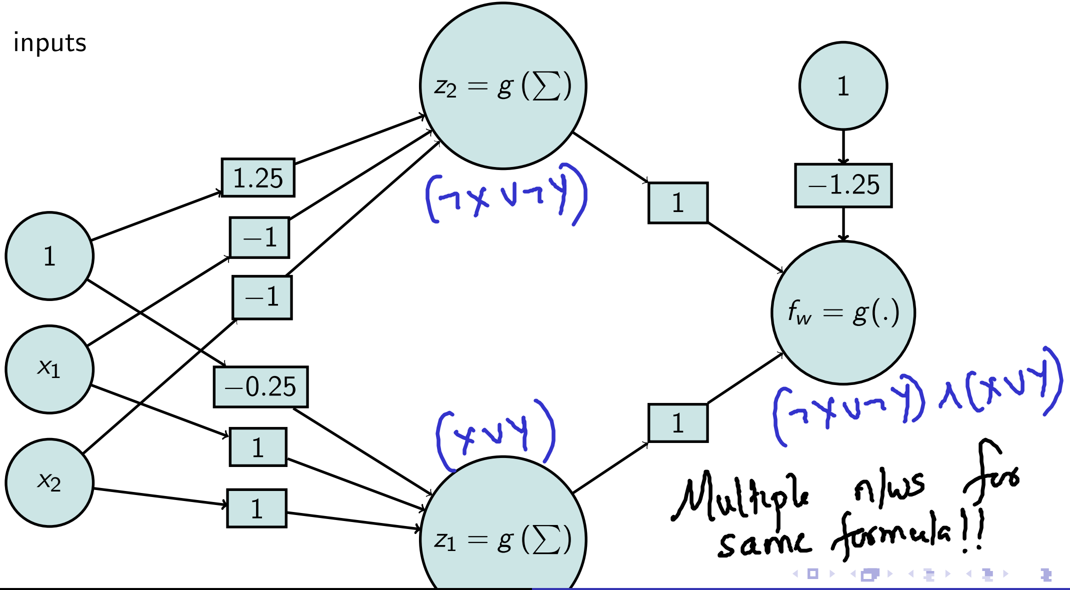
$(x \wedge \bar{y}) \vee (\bar{x} \wedge y)$

Recall: XOR was not linearly seperable (discussion on VC dimensions)

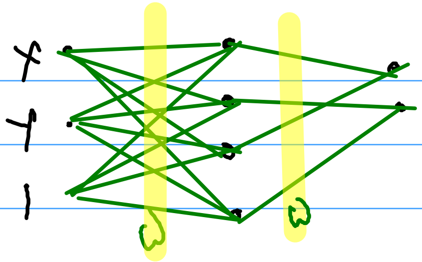
How about fewer hidden layers for XOR?

Ans: $x \wedge \bar{y}$ & $\bar{x} \wedge y$ are representable using linear seperator!

Eg: Feed-forward Neural Net for XOR ($\theta = 0$)



For XOR & XNOR in same n/w



$(\bar{x} \wedge y) \vee (x \wedge \bar{y})$ [XOR]

$(\bar{x} \wedge \bar{y}) \vee (x \wedge y)$ [XNOR]

Q: How to "train" the network (ie learn the w's)?

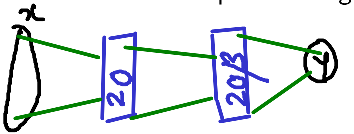
ϕ is left to be x (trivial ϕ)

STEP 0: Pick a network architecture

- Number of input units: Dimension of features $\phi\left(\mathbf{x}^{(i)}\right)$.
- Number of output units: Number of classes.
- 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.
- We will use

  Hope that hidden layers represent "true" ϕ.

  - 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** (using [stochastic] gradient desc)
  - instead of the non-smooth step function $g(s) = 1$ if $s \in [\theta, \infty)$ and $g(s) = 0$ otherwise.



Tutorial 7, problem 5

High Level Overview of Backpropagation Algorithm for Training NN

1. Randomly initialize weights w_{ij}^{l} 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 $x \in \mathcal{D}$.
3. Execute **backpropagation**

   error correction

   1. by computing partial derivatives $\frac{\partial}{\partial w_{ij}^{(l)}} E(w)$ 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 $\mathbf{w}$.

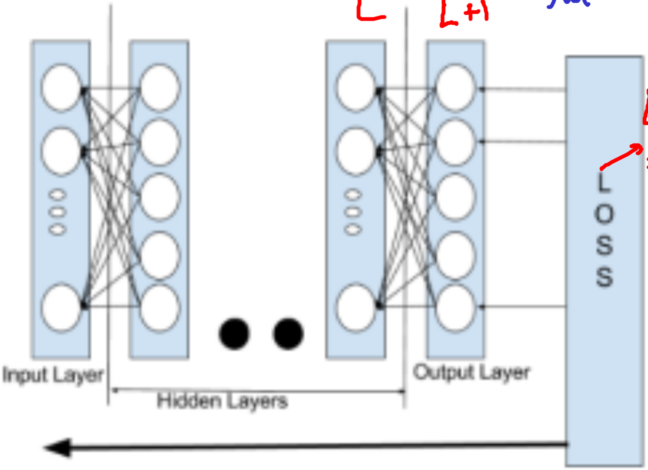
$$w_{ij}^{l} = w_{ij}^{l} - \eta \frac{\partial}{\partial w_{ij}^{(l)}} E(w)$$

Key step

4. Verify that the cost function $E(w)$ has indeed reduced, else resort to some random perturbation of weights $\mathbf{w}$.

$$f_w(x^i) = sig\left(\sum_{i=1}^{s_L} w_i^{(L)} sig_j^{L}\right)$$

L $\quad$ $L+1$



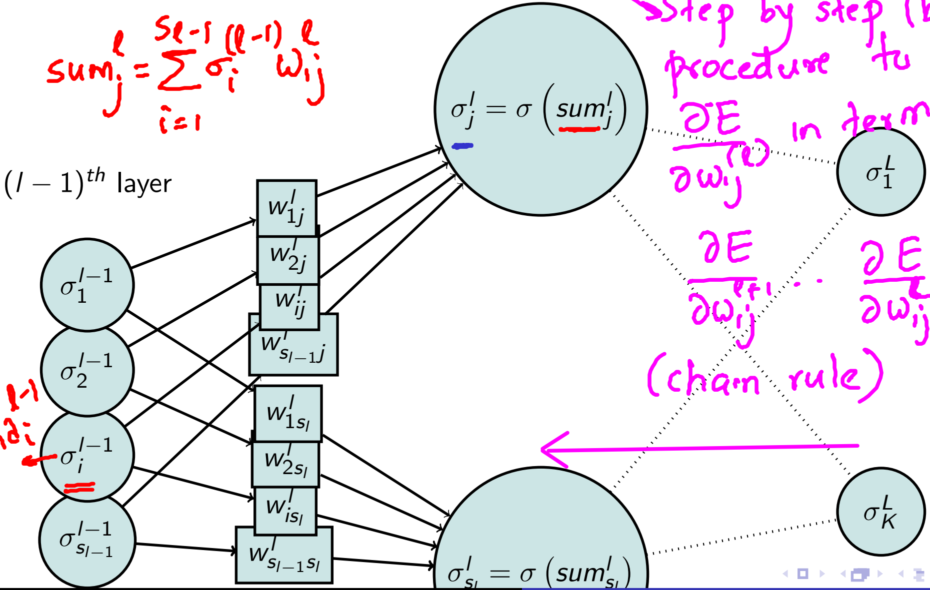
$$E(w) = -\sum_i y^i \log f_w(x^i) + (1-y^i)\log(1-f_w(x^i)) + \sum_l \sum_i \sum_j |w_{ij}^l|^2.$$

Iteratively update the weights along the direction where Loss decreases.

Setting Notation for Backpropagation

$$sum_j^l = \sum_{i=1}^{s_{l-1}} \sigma_i^{(l-1)} w_{ij}^l$$

Step by step (backward) procedure to obtain $\frac{\partial E}{\partial w_{ij}^{(l)}}$ in terms of $\frac{\partial E}{\partial w_{ij}^{l+1}} \cdots \frac{\partial E}{\partial w_{ij}^{L}}$ (chain rule)



Gradient Computation

- The Neural Network objective to be minimized:

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

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

easy to handle

$$\frac{\partial E(w)}{\partial w_{ij}^{(l)}} = \sum_{i=1}^{m}\sum_{k=1}^{K}\frac{\partial E(w)}{\partial \sigma_k^L(x^i)}\sum_{j=1}^{s_{L-1}}\frac{\partial \sigma_k^L(x^i)}{\partial sum_j^{L-1}(x^i)}\sum_{p=1}^{s_{L-1}}\frac{\partial sum_j^{L-1}}{\partial \sigma_p^{L-1}} \cdots \frac{\partial sum_j^{l}(x^i)}{\partial w_{ij}^{(l)}}$$

Gradient Computation

- The Neural Network objective to be minimized:

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

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} \frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} = \left(\frac{1}{1+e^{-sum_i^l}}\right)\left(1-\frac{1}{1+e^{-sum_i^l}}\right) = \sigma_\mathbf{j}^\mathbf{l}(\mathbf{1}-\sigma_\mathbf{j}^\mathbf{l})$
- $\frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial}{\partial w_{ij}^l}\left(\sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}\right) = \sigma_\mathbf{i}^\mathbf{l-1}$